

**ÉCOLE DE TECHNOLOGIE SUPÉRIEURE  
UNIVERSITÉ DU QUÉBEC**

**THESIS PRESENTED TO  
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE**

**IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR  
THE DEGREE OF DOCTOR OF PHILOSOPHY  
Ph.D.**

**BY  
Eduardo MIRANDA**

**IMPROVING THE ESTIMATION, CONTINGENCY PLANNING AND TRACKING OF  
AGILE SOFTWARE DEVELOPMENT PROJECTS**

**MONTREAL, AUGUST 26, 2010**

© Copyright 2010 reserved by Eduardo Miranda

**BOARD OF EXAMINERS**

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

M. Pierre Bourque, directeur de thèse

Département de génie logiciel et des technologies de l'information à l'École de technologie supérieure

M. Alain Abran, codirecteur de thèse

Département de génie logiciel et des technologies de l'information à l'École de technologie supérieure

M. Jean-Pierre Kenné, président du jury

Département de génie mécanique à l'École de technologie supérieure

M. Yann-Gaël Guéhenec, examinateur externe

Département de génie informatique et génie logiciel, École polytechnique de Montréal

THIS THESIS WAS PRESENTED AND DEFENDED

BEFORE A BOARD OF EXAMINERS AND PUBLIC

AUGUST 25, 2010

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

## **ACKNOWLEDGEMENTS**

To my wife Mariana and my son Tomás for all their encouragement and patient. To my thesis directors Dr. Pierre Bourque and Dr. Alain Abran for their advice and their time. To Dr. Alain April and Dr. Serghei Floricel for their support during difficult times. To all my friends.

It is impossible to know where life would have taken us have we taken that decision instead of this one, but when we look back, it is very clear how one thing led to another. Getting here took a lot of effort and courage, but it would have not been possible without you all.

Merci,

# **IMPROVING THE ESTIMATION, CONTINGENCY PLANNING AND TRACKING OF AGILE SOFTWARE DEVELOPMENT PROJECTS**

Eduardo MIRANDA

## **RÉSUMÉ**

Les progrès technologiques dus aux logiciels présents dans nos maisons, nos voitures et nos téléphones ont également apporté des changements à la manière dont cette même technologie est développée. L'augmentation de la puissance des ordinateurs a permis l'émergence d'une nouvelle approche pour développer du logiciel. Cette approche s'appuie sur des composants, du prototypage et des cycles de développement courts, plutôt que sur une approche plus traditionnelle - mais pas si lointaine - basée sur des phases d'analyse, de conception et de réalisation. Cette nouvelle approche s'intitule le développement agile. Cette thèse porte sur trois aspects distincts, bien qu'étroitement liés, de la gestion de ce type de projets dits agiles de développement de logiciel, à savoir :

- l'estimation de la taille du logiciel pour planifier un projet,
- le suivi des activités de développement et,
- le calcul et l'administration des fonds de réserve

La taille du logiciel constitue l'intrant principal au processus d'estimation de l'effort et de la durée d'un projet de développement de logiciel. En conséquence, déterminer des évaluations crédibles et fiables de cette taille est primordial au processus d'estimation. Cette thèse propose en premier l'utilisation d'une méthode modifiée de comparaison par paires pour appuyer le jugement des experts - méthode d'estimation la plus utilisée dans l'industrie du logiciel. Dans la méthode modifiée proposée, le nombre de comparaisons, un facteur limitant l'utilisation de la méthode à grande échelle, est réduit presque de moitié par l'emploi des designs cycliques incomplets (Incomplete Cyclic Design - ICD) pour choisir des paires appropriées d'entités à comparer.

Cette thèse porte en deuxième sur le suivi d'un projet qui consiste en un processus de comparaison de l'avancement du projet anticipé selon la planification par rapport à son avancement réel, afin de décider, s'il y a lieu, des actions nécessaires pour le compléter tel

que prévu. La thèse propose et montre l'utilisation d'un indicateur de ligne de mise à jour (Line of Balance - LOB) modifié en vue d'obtenir des informations non disponibles avec les « burn down charts » et les diagrammes de flux cumulatif (cumulative flow diagram), les deux indicateurs les plus fréquemment utilisés dans les projets dits agiles. La contribution de la thèse s'inscrit, non seulement en termes de la nouveauté de l'application de cet indicateur LOB aux projets de développement de logiciels, mais également dans le remplacement des calculs de délai provenant des plans originaux, par de l'information extraite directement d'un système de contrôle de versions.

Cette thèse porte en troisième lieu sur les fonds de réserve tels que définis par le Project Management Institute (PMI) comme la quantité de fonds requis au-delà de l'estimation pour ramener le risque de dépassements à un niveau acceptable pour l'organisation. Cette thèse postule qu'un calcul réaliste de ces fonds devrait être basé sur le coût de maintien du projet dans les délais, et non sur ce qu'aurait coûté ce travail s'il avait été prévu dès le commencement. Cette thèse propose en outre un modèle quantitatif tenant compte de la taille du projet, du moment auquel la sous-estimation a été reconnue et des pertes de processus liées aux actions de rétablissement. Les résultats du modèle permettent l'exploration des coûts et des avantages de plusieurs alternatives de gestion.

Les trois méthodes présentées s'avéreront du plus grand intérêt pour les chefs de projet, les ingénieurs logiciel et les autres intervenants impliqués dans la planification des projets et les activités de gestion des risques. Même si les exemples employés pour illustrer et expliquer les concepts correspondent aux projets utilisant des approches agiles telles que « Scrum » et « Feature Driven Development », les méthodes proposées s'appliquent aussi aux autres types de développement de logiciels.

**Mots-clés:** estimation du logiciel, suivi des activités, ligne de mise à jour, calcul et administration des fonds de réserve, développement agile, comparaison par paires

# **IMPROVING THE ESTIMATION, CONTINGENCY PLANNING AND TRACKING OF AGILE SOFTWARE DEVELOPMENT PROJECTS**

Eduardo MIRANDA

## **ABSTRACT**

The technological advances that have brought us computers in our homes, our cars and our telephones have also brought about changes to the way that very same technology is developed. The abundance of computer power has enabled a new way of developing software that relies on components, prototyping and short development cycles, rather than on the more traditional analysis, design and build phases of not that long ago. This new way of developing software is called Agile development. This research looks into three distinct, but related, aspects of the management of Agile projects: (1) estimating software size with the purpose of planning a project, (2) monitoring development activities, and (3) calculating and administering contingency funds, and proposes new methods for addressing them.

Software sizing provides the foundation for estimating effort and project duration, and so the importance of credible and reliable size estimates cannot be overstated. To address the issue of estimation, the thesis proposes a modified Paired Comparison method to support expert judgement, the prevailing sizing method used in industry. In this method, the total number of comparisons, which is a factor limiting the scalability of the method, is reduced almost by half using incomplete cyclic designs (ICD) to select suitable pairs of entities to be compared.

Monitoring a project, is the process of comparing how far it has come relative to where it was supposed to be according to its plan, for the purpose of deciding what, if any, actions are necessary to complete it as planned. This thesis proposes and demonstrates the use of a modified line of balance (LOB) indicator to gain insights into the project's progress not provided by burn-down charts and cumulative flow diagrams, the two most common indicators used in Agile projects. The contribution of the thesis can be measured not only in terms of the novelty of the application of the LOB indicator to software development

projects, but also in the replacement of the original plan-based lead-time calculations with dynamic information extracted from a control version system.

Contingency is defined by the Project Management Institute as the amount of funds needed above the estimate to reduce the risk of overruns to a level acceptable to the organization. This thesis postulates: 1) that a realistic calculation of these funds should be based on the cost of keeping the project on-schedule, and not on what it would have cost had the work been planned from the beginning, and proposes a quantitative model which takes into account the size of the project, the time at which the underestimation is acknowledged and the process losses associated with the recovery actions, and 2) that these funds ought to be administered above the project level to preserve the premise that their use is probabilistic. The model's outputs enable the exploration of the costs and benefits of several management options.

The three methods presented will be of interest to project managers, software engineers and others involved in planning and risk management activities. While the examples used to illustrate and explain the concepts correspond to projects using Agile approaches, such as Scrum and Feature-Driven Development, the methods proposed are applicable to other types of development as well.

**Key words:** software estimation, project tracking, line of balance, calculation of contingency funds, agile development, paired comparisons

## TABLE OF CONTENTS

	Page
INTRODUCTION .....	1
CHAPTER 1 CURRENT CHALLENGES IN THE MANAGEMENT OF SOFTWARE DEVELOPMENT PROJECTS .....	3
1.1 A Tally of Successes and Failures .....	3
1.2 Incremental Development .....	4
1.3 Estimating the Effort Required .....	6
1.4 Determining how much has been accomplished .....	9
1.5 The underestimation of contingency funds .....	11
CHAPTER 2 AGILE DEVELOPMENT .....	15
2.1 A retrospective on Agile Development .....	15
2.2 Feature-Driven Development .....	19
2.3 Scrum .....	21
CHAPTER 3 MAJOR THEMES .....	24
3.1 Sizing User Stories Using Paired Comparisons .....	24
3.2 Protecting Software Development Projects against Underestimation .....	25
3.3 Agile Monitoring Using the Line of Balance .....	27
3.4 The Articles in Context .....	28
3.4.1 The Guide to the Software Engineering Body of Knowledge .....	28
3.4.2 Traceability between the Publications and the SWEBOK .....	29
CONCLUSION .....	31
ANNEX I .....	34
Sizing User Stories Using Paired Comparisons .....	34
ANNEX II .....	46
Protecting Software Development Projects against Underestimation .....	46
ANNEX III .....	60
Agile Monitoring Using the Line of Balance .....	60
APPENDIX A .....	73
Scrum roles, artifacts and management practices .....	73
APPENDIX B .....	77



Definition of relevant topics and subtopics in the Guide to the SWEBOK.....	77
LIST OF BIBLIOGRAPHICAL REFERENCES.....	79

## LIST OF TABLES

	Page
Table 1 Results of cost overrun surveys .....	4
Table 2 Work Allocation in Feature-Driven Development Projects .....	20
Table 3 SWEBOK – Publications Traceability Matrix.....	30
Table 4 Scrum Roles .....	73
Table 5 Scrum Artifacts .....	74
Table 6 Scrum Management Practices.....	75

## LIST OF ILLUSTRATIONS

	Page
Figure 1 Project success trends according to the Standish Group. ....	3
Figure 2 (a) All-at-once development vs. (b) incremental development .....	5
Figure 3 Correlation between final functionality and schedule estimation error.....	7
Figure 4 Size estimations of the same system using different approaches: .....	9
Figure 5 The Freiman curve.....	11
Figure 6 Relative importance assigned to various project success factors .....	12
Figure 7 Relative cost of fixing a defect.....	18
Figure 8 Feature-Driven Development process .....	19
Figure 9 Diffusion of various software development processes,.....	22
Figure 10 The Scrum process .....	23
Figure 11 Software Engineering Management Knowledge Area topics .....	29

## **LIST OF ABBREVIATIONS AND ACRONYMS**

FDD	Feature-Driven Development
ICD	Incomplete Cyclic Design
LOB	Line of Balance
PMI	Project Management Institute
SWEBOK	Software Engineering Body of Knowledge
XP	Extreme Programming

## INTRODUCTION

Over the last ten years, the author has proposed a number of methods that address endemic problems in the area of software project management, such as estimation, planning and controlling. These methods have evolved from the author's extensive industrial experience, and their usefulness has become established in many projects at Ericsson, the author's former employer. The techniques in question include the Paired Comparison estimation method (Miranda 2001a), the statistically planned Incremental Development method (Miranda 2002), the Rate of Growth Monitoring method (Miranda 1998), the line-of-balance (LOB) indicator for tracking progress (Miranda 2006) and a project screening method (Miranda 2001b). Building on this foundation, this thesis proposes three new methods to support decision-making by industry practitioners in general, and by projects using Agile approaches in particular.

The thesis consists of three related peer reviewed journal publications: Sizing User Stories Using Paired Comparisons (Miranda, Bourque et al. 2009), Protecting Software Development Projects Against Underestimation (Miranda and Abran 2008) and Agile Monitoring Using the Line of Balance (Miranda and Bourque 2010). This thesis includes also the complementary information necessary to ensure global comprehension and the linkage between the various parts.

While the examples illustrating and explaining concepts throughout the thesis have been drawn from the Scrum (Rising and Janoff 2000; Schwaber and Beedle 2004; Cohn 2006) and Feature-Driven Development (Coad, Lefebvre et al. 1999; Palmer and Felsing 2002; Anderson 2004) processes, the proposed methods are applicable to other types of development as well.

The thesis is organized as follows: CHAPTER 1 provides a literature review on the current challenges in the management of software development projects in general, and Agile projects in particular. CHAPTER 2 includes an overview of Agile development and its

underlying assumptions, as well as an introduction to Feature-Driven Development and Scrum to give the reader two concrete examples of Agile processes. CHAPTER 3 provides an overview of each publication's findings and contributions. The CONCLUSION closes the thesis by summarizing the work done and proposing new topics for research. The actual journal publications are included as annexes in the format in which they were originally published in the refereed journals.

## CHAPTER 1

### CURRENT CHALLENGES IN THE MANAGEMENT OF SOFTWARE DEVELOPMENT PROJECTS

#### 1.1 A Tally of Successes and Failures

Figure 1, constructed using the data from the Standish Group Chaos Reports cited by Eveleens and Verhoef (Eveleens and Verhoef 2010) shows, that since the publication of the first report in 1995 (Standish Group 1995), the software development community has been making significant progress in its ability to complete development projects on-time and on-budget. Despite this progress, a large number of projects still finish late and over budget. A study by El Emam and Koru (El Emam and Koru 2008) puts the average number of challenged and failed projects at 50%, while the meta analysis of Jorgensen and Molokken (Jorgensen and Molokken 2006) puts it at about 33% (see Table 1).

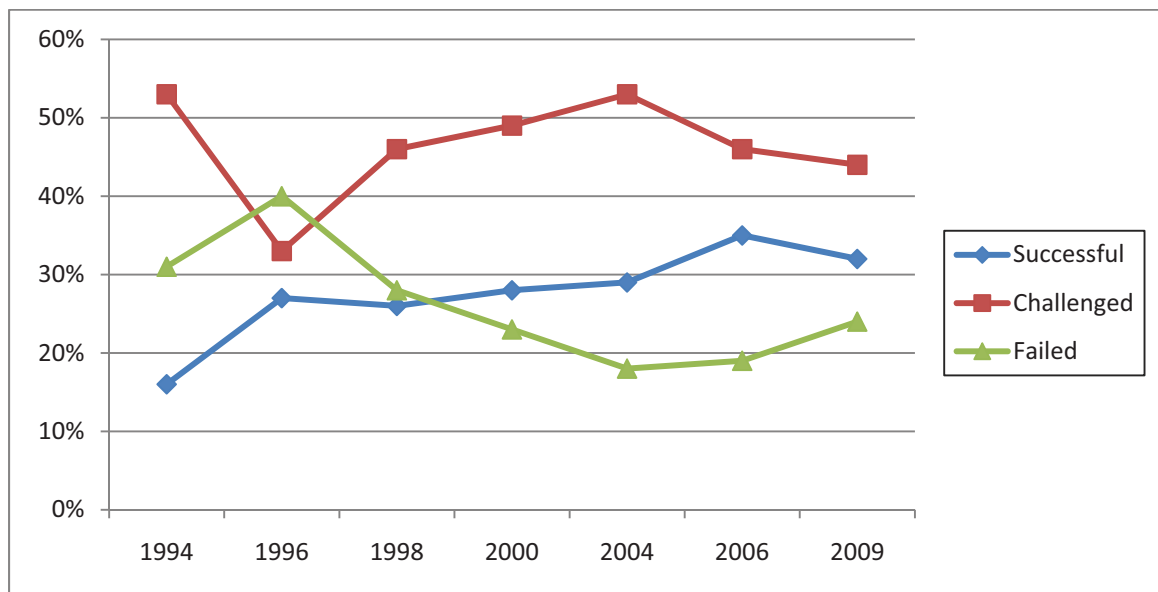


Figure 1 Project success trends according to the Standish Group.

Table 1 Results of cost overrun surveys  
(After Jorgensen and Molokken 2006)<sup>1</sup>

Study	Jenkins	Phan	Bergeron
Year	1984	1988	1992
Size of the study	23 software organizations	191 software projects	89 software projects
Country of respondents	USA	USA	Canada
Average cost overrun	34%	33%	33%

## 1.2 Incremental Development

Coincident with this improvement in project predictability there has been a move away from the monolithic or waterfall model of development, pointed out in the 1994 CHAOS Report (Standish Group 1995) as one of the root causes of the high failure rate, towards an incremental<sup>2</sup>, fine grained development paradigm in which the goal, or the total software system capability, is achieved by breaking down that goal into smaller, disjoint sub-goals,

---

<sup>1</sup> Used with permission

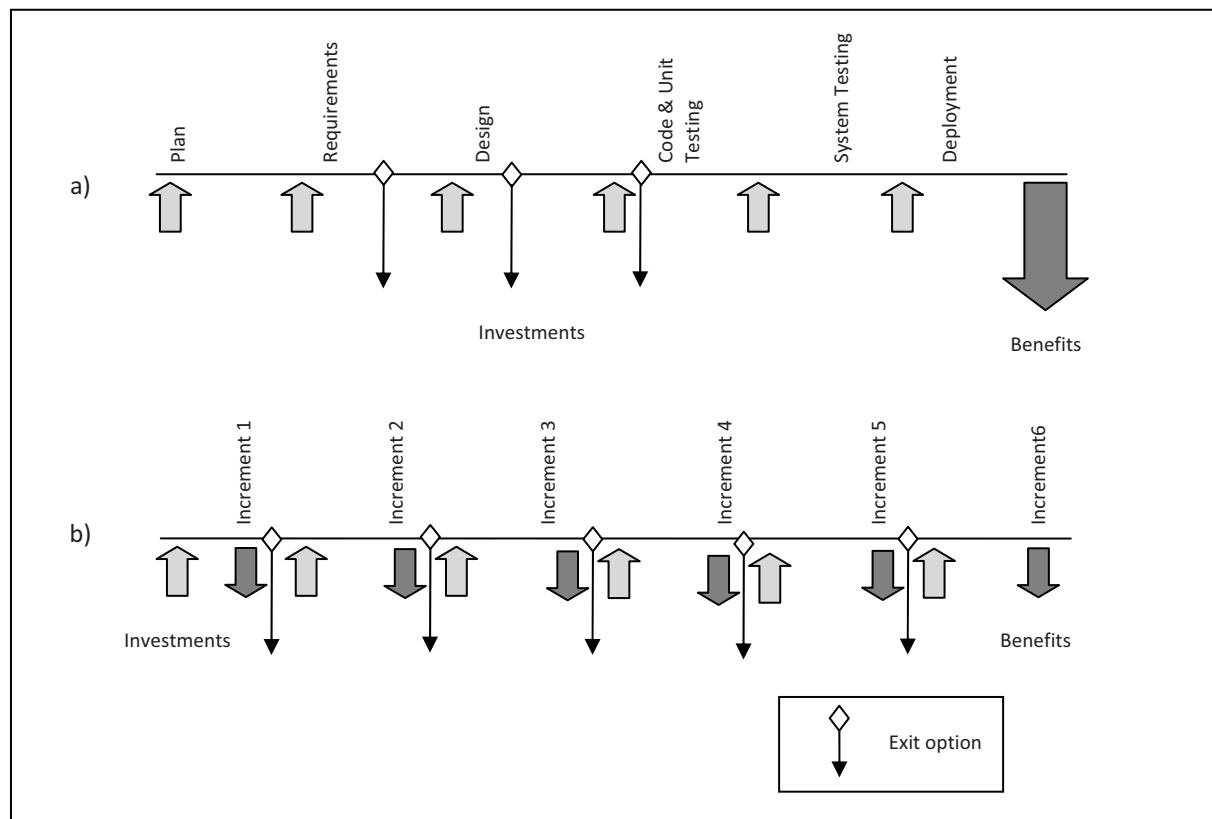
<sup>2</sup> Incremental development should not be confused with iterative development. The purpose of the former is to break down the total scope of a project into a number of functioning, albeit incomplete, systems, while the purpose of the latter is to learn and refine the scope of the system through successive cycles of analysis, development and testing.



and implementing them in a phased approach. This is exemplified by the recommendations for projects with “smaller milestones” (Standish Group 1995; Standish Group 1999), and projects with “minimized scope” (Standish Group 2001). Figure 2 compares the two types of development.

By making available a critical core of functionality as early as possible, incremental development enables, among other things, the following (Denne and Cleland-Huang 2004; Erdogmus 2005):

- Smaller projects which are easier to plan and control
- Earlier revenue generation
- Postponement of unclear/uncertain decisions



**Figure 2 (a) All-at-once development vs. (b) incremental development (After Erdogmus 2005).**

Incremental development is not a new idea. In one form or another, it can be traced back to the 1970s (Larman and Basili 2003). In 1980, Mills (Mills 1980) reported the successful completion of the US Navy project, LAMPS, using incremental development to deliver 7 million lines of code in 45 increments. This 200 person-year project was completed in four years with monthly deliveries, and completed on time and within budget.

However, the monolithic, all-or-nothing type of approach mentioned above is by no means the sole cause of project failure. Among the many reasons explaining project failure in the studies conducted by Pinto and Mantel (Pinto and Mantel 1990), Shenhar et al. (Shenhar, Dvir et al. 2001) and Drew Procaccineo et al. (Drew Procaccino, Verner et al. 2002), three seem to be prevalent in the software engineering industry: a lack of reliable size measures leading to realistic schedule and effort estimates (Linberg 1999), an inability to understand the degree of progress (Rozenes, Vitner et al. 2006) and how ill-conceived estimates to recover a project based on the past consumption of resources (Lipke 1999; Defense 2003) fall short of what is necessary to produce the desired turnaround, and mostly result in wasted effort (Keil and Mann 1997).

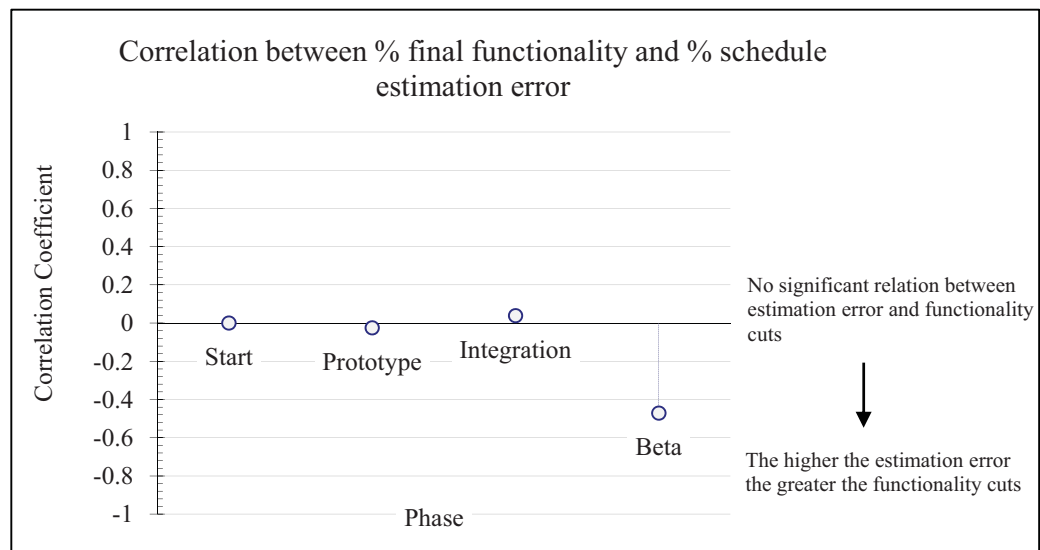
### **1.3 Estimating the Effort Required**

The negative correlation between final functionality and estimation error (see Figure 3), based on data from Upadhyayula (Upadhyayula 2001), shows how HP-Agilent reacted to over-commitment in a sample of twenty-two projects. Estimation errors are introduced early in a project, since estimation is one of the first activities to take place. The data, however, show no acknowledgment of any underestimation via a reduction in scope through the life of the project. Only as the committed target date for delivery approached, or was left behind, did schedule pressure mount and was there a cut in functionality. This behaviour, which results in stakeholder frustration and wasted effort, is not unique to HP-Agilent (PRTM 1995). Part of the problem is the lack of quantitative techniques for estimating, planning and controlling incremental projects, as exemplified by the heuristics used at Microsoft:

“divide the project in three subprojects and allocate the first 1/3 of features (the most critical features and shared components) to the first subproject, the second 1/3 of features to the second subproject and the final 1/3 of features (the least critical) to the third subproject” (Cusumano and Selby 1997)

or by the “planning game” in Extreme Programming:

“the planning game begins when the customer starts writing stories. They write a card, the programmers look at the story and try to estimate it, but they come back and say “it’s too big.” The customer then says, “Let me split it into a few cards,” writes some new ones, and throws the original away” (Wake 2001)



**Figure 3 Correlation between final functionality and schedule estimation error. Data from HP-Agilent (After Upadhyayula 2001).**

Figure 4 displays the results of two experiments on the reliability<sup>3</sup> of size estimations conducted by the author while working at Ericsson<sup>4</sup> and teaching at Carnegie Mellon

---

<sup>3</sup> The reliability of an estimate includes its accuracy (closeness of agreement between a measured quantity value and a true quantity value of a measure) and precision (closeness of agreement between indications or measured

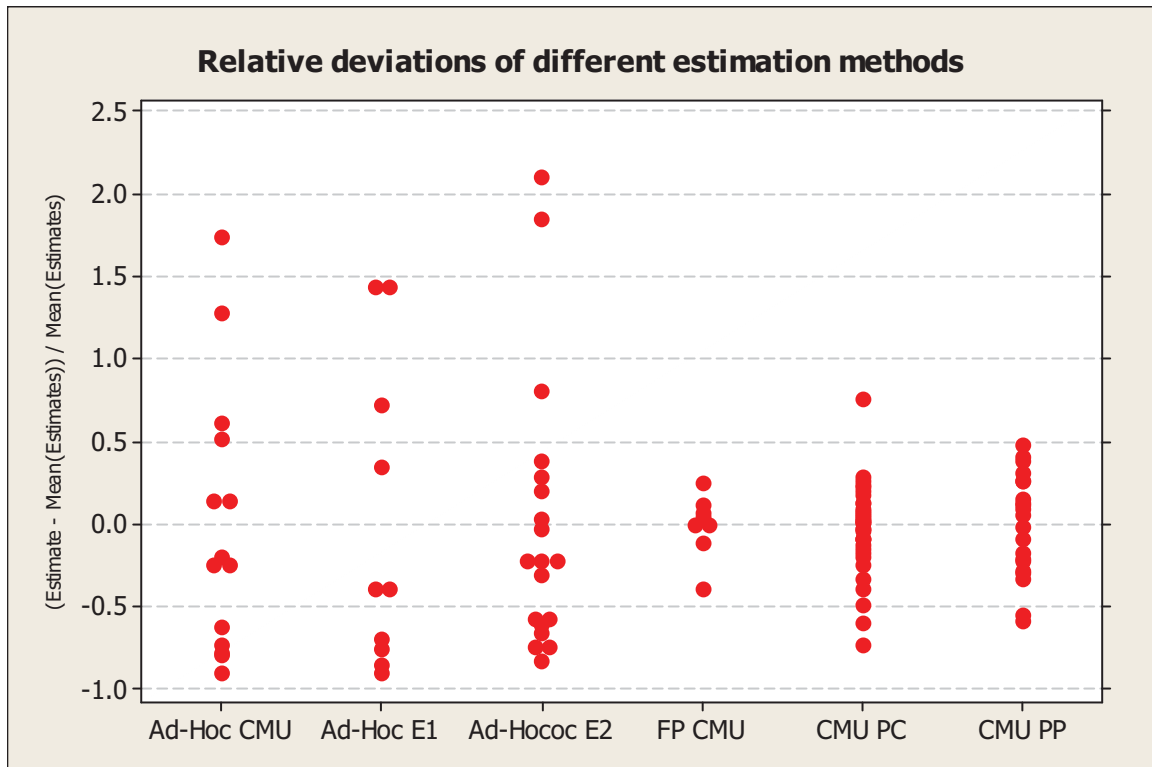
University, comparing the results of unstructured approaches such as the one proposed by Wake (Wake 2001) to those of more structured methods such as Function Point Analysis (UKSMA 1998; NESMA 2002; IFPUG 2006 ), Planning Poker (Grenning 2002; Cohn 2006) and Paired Comparisons (Bozoki 1993; Shepperd and Cartwright 2001; Miranda 2001a). Similar findings have been reported by Sasao (Sasao 2009), and are consistent with the reliability of Function Point counts, published by Kemerer (Kemerer 1993). The chart clearly shows that, in the absence of guidelines and reference points, estimates lack reliability. The ad hoc estimates, labelled “Ad Hoc CMU”, “Ad Hoc E1” and “Ad Hoc E2”, exhibit greater variance than Function Point counting, Planning Poker and Paired Comparisons, labelled “FP-CMU”, “PP CMU” and “PC CMU” respectively.

If it is assumed, as in auction theory (Thaler 1988; Connolly and Dean 1997; Svendsgaard 2004; Jorgensen and Grimstad 2005), that the average of the estimations (normalized to correspond to level 0 of the chart in Figure 4) represents a feasible estimate for the project, then approximately half the projects would have been underestimated and would have had to be re-planned midway through their execution, usually at a higher cost, as explained in (Miranda and Abran 2008), while the other half or so of the projects would have been overestimated. However, as explained by thanks to Parkinson’s Law (Parkinson 1955; Gutierrez and Kouvelis 1991; Brannon, Hershberger et al. 1999; Kujawski, Alvaro et al. 2004), the savings would never have materialized, as work expanded until the budget was exhausted. This tendency of costs to increase as they move beyond the optimal level (see Figure 5) was also noted by Freiman (Freiman 1983) in 1983.

---

quantity values obtained by replicate measurements on the same or similar objects under specified conditions). ISO/IEC (2007). International vocabulary of metrology — Basic and general concepts and associated terms (VIM). Geneva, ISO/IEC.

<sup>4</sup> The author has observed similar patterns when teaching estimation courses in a variety of settings.



**Figure 4 Size estimations of the same system using different approaches: Ad hoc, Function Point counting, Paired Comparisons and Planning Poker.**

#### 1.4 Determining how much has been accomplished

Estimates, and, by extension, the plans they support, are based on assumptions that must be constantly monitored, so that, when deviations between what was assumed and reality materializes, appropriate corrective actions can be taken to re-establish the course or at least bring it to the next most desirable state.

Established project monitoring methods, such as earned value (Project Management Institute 2004) and bug convergence (Lory, Campbell et al. 2003), are based on what has already happened and are more oriented towards external reporting than to providing early warning signals which could be used to hypothesize or predict what may happen in the near future, when there is still time to do something about it (Miranda 1998).

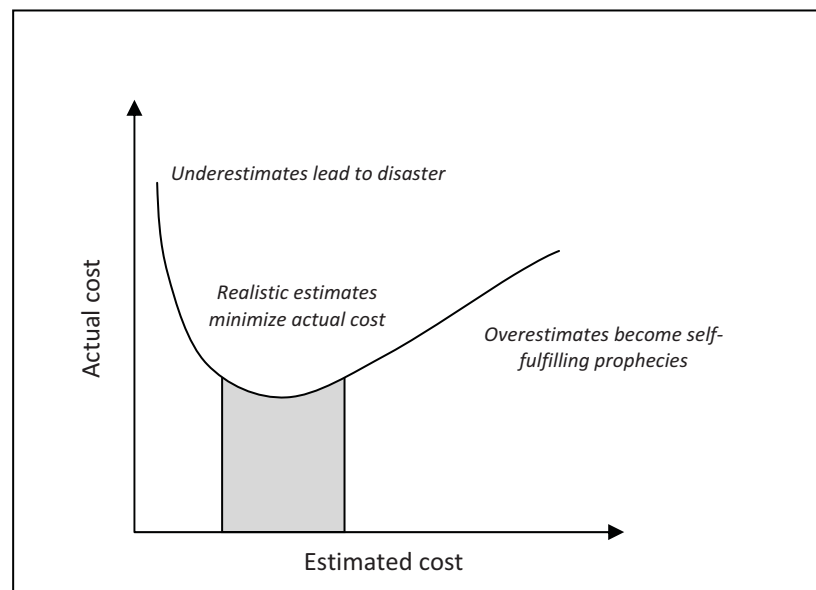
According to Balachandra (Balachandra 1989), early warning indicators should have the following characteristics:

- They should be easily measurable;
- They should be capable of being consistently represented and measured by different evaluators;
- They should pertain to the project itself and not measure conditions that affect all the projects within the company.

An implementation of these three characteristics can be found in the Practical Software and Systems Measurement Guide (McGarry and Jones 2004), where project issues are differentiated from organizational issues and used to structure a measurement plan for the project.

To the three characteristics of early warning indicators listed above two more could be added: 1) early warning indicators should minimize false alarms, and 2) they should give ample time to react. These two requirements conflict, however, since the longer the anticipatory time, the weaker the signals of trouble and the higher the chance of false positives. Examples of early warning indicators include the use of reliability growth models (Miranda 1998; Staron and Meding 2007), code churn (Towell and Denton 2006) and technical performance measurement indicators (Coleman, Kulick et al. 1996; INCOSE 2004).

Monitoring progress requires information not only information about what has been completed, but also about work in progress and the rate of progress (Miranda and Bourque 2009). In this way, appropriate responses can be planned and resources directed to where they are needed, since nothing is gained by improving non-constraining resources in a process (Goldratt 2004).

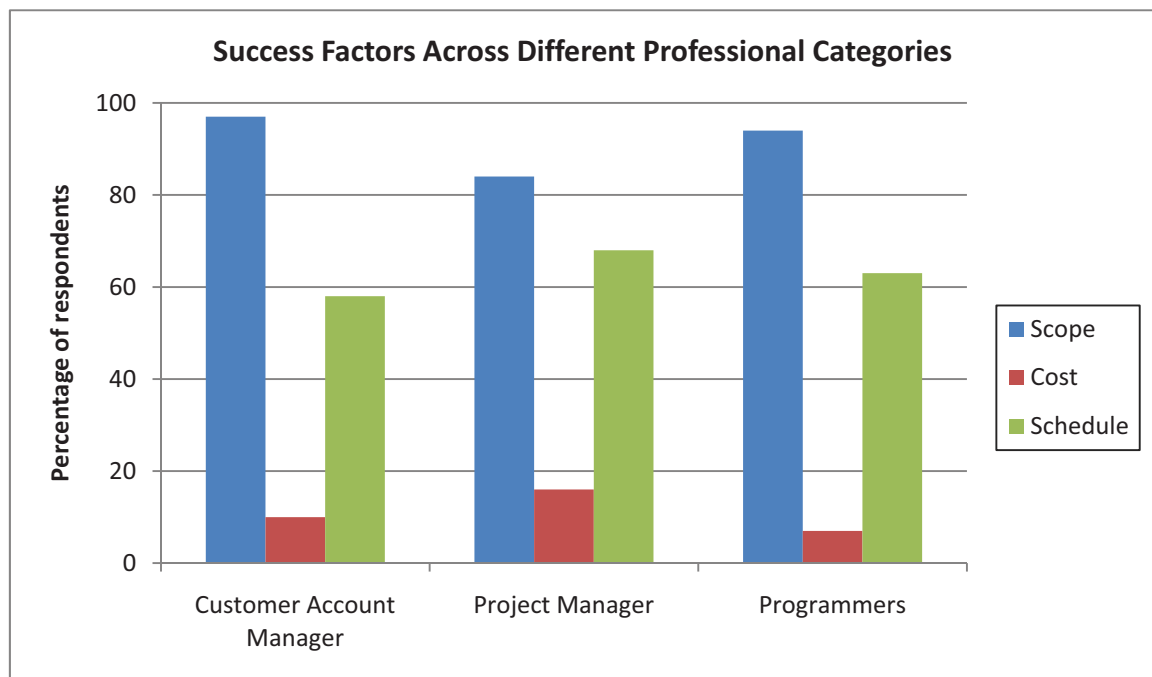


**Figure 5 The Freiman curve  
(After Freiman 1983).**

### **1.5 The underestimation of contingency funds**

While there is no doubt that project success is a multidimensional social construct (Tishler, Dvir et al. 1996; Shenhar, Dvir et al. 2001; Smith-Doerr, Manev et al. 2004; Thomas and Fernández 2008), the survey by Agarwal and Rathod (Agarwal and Rathod 2006) of 105 software professionals clearly shows that some dimensions (see Figure 6) are consistently valued more highly than others by customer managers, project managers and developers.

The preference for maintaining scope and meeting schedule is reinforced by a passage from the report on the accident of the Columbia space shuttle:



**Figure 6 Relative importance assigned to various project success factors (After Agarwal and Rathod 2006)<sup>5</sup>.**

“During the course of this investigation, the Board received several unsolicited comments from NASA personnel regarding pressure to meet a schedule. These comments all concerned a date, more than a year after the launch of Columbia, that seemed etched in stone: February 19, 2004, the scheduled launch date of STS-120. This flight was a milestone in the minds of NASA management since it would carry a section of the International Space Station called “Node 2.” ... At first glance, the Core Complete configuration date seemed noteworthy but unrelated to the Columbia accident. However, as the investigation continued, it became apparent that the complexity and political mandates surrounding the International Space Station Program, as well as Shuttle Program management’s responses to them, resulted in pressure to meet an increasingly ambitious launch schedule.

---

<sup>5</sup> Used with permission



In mid-2001, NASA adopted plans to make the over-budget and behind-schedule International Space Station credible to the White House and Congress. The Space Station Program and NASA were on probation, and had to prove they could meet schedules and budgets. The plan to regain credibility focused on the February 19, 2004, date for the launch of Node 2 and the resultant Core Complete status. If this goal was not met, NASA would risk losing support from the White House and Congress for subsequent Space Station growth”. (Columbia Accident Investigation Board 2003)

In a similar vein, in his book, *Practical Risk Assessment for Project Management*, Grey wrote:

“While most people are willing to accept that cost could exceed expectations, and might even take a perverse delight in recounting past examples, the same is not true of deadlines. This is probably due to the fact that cost overruns are resolved in-house, while schedule issues are open and visible to the customer.” (Grey 1995)

While the quotes above can be regarded as anecdotal, they contribute to generalize the preference of schedule over cost reported by Agarwal and Rathod (Agarwal and Rathod 2006).

Austin (Austin 2001) and Kujawski (Kujawski, Alvaro et al. 2004) argue that the repercussions of project success in terms of career progression cannot be ignored in the modeling of real-world decision-making. This, and the fact that the cost of recovering from a schedule slip is higher than the cost of doing the work as originally planned (Cooper 1994; Hsia, Hsu et al. 1999), suggests that the size of any contingency fund must be calculated at recovery level, and not as the cost to perform the work had it been planned from the beginning.

This chapter has presented three important challenges facing most projects: the reliability of the estimates, the ability to determine the state of advancement in a timely fashion and the need to incorporate the human and organizational considerations that heavily influence decision-making in the real world into the calculation of contingency funds. The next chapter introduces the Agile development approach.

## CHAPTER 2

### AGILE DEVELOPMENT

#### 2.1 A retrospective on Agile Development

In the mid-1990s, a new paradigm for software development began to emerge. Abundant desktop computing power and communication bandwidth, which allowed rapid feedback through application composition, shorter iterations, frequent builds and continuous integration (Dullemond, Gameraen et al. 2009), shifted software development away from plans and design documents as coordination mechanisms towards code and daily meetings (Raymond 2000). From a business perspective, shorter product life spans, 12 to 18 months in the mobile device industry (Aramand 2008) and less than a year in the Internet world (Barksdale and Berman 2007), blurred the distinction between development and maintenance into something more akin to continuous development (Dalcher 2003), making difficult to justify the need for resilient architectures and the preparation of support documentations with respect to what it was at the beginning of the '90s, when the average application lifespan was 9 years (Tamai and Torimitsu 1992). In this regard Beck (Beck 2000) states that “Maintenance is really the normal state of an XP project”.

These technical and business conditions materialized in a number of lightweight processes collectively referred to as “agile”. In 2001, seventeen consultants and practitioners entered into a consortium and published a document, titled “The Agile Manifesto” (Beck, Beedle et al. 2001), setting out what they referred to as “the principles” behind the new approach:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software;
2. (we) Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage;
3. (we) Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale;
4. Business people and developers must work together daily throughout the project;
5. (we) Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done;
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation;
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely;
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity - the art of maximizing the amount of work not done - is essential;
11. The best architectures, requirements, and designs emerge from self-organizing teams;
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

A non-exhaustive list of Agile processes includes Scrum (Beedle, Devos et al. 2000; Rising and Janoff 2000; Schwaber and Beedle 2004; Cohn 2006), Feature-Driven Development (Coad, Lefebvre et al. 1999; Palmer and Felsing 2002; Anderson 2004), Extreme Programming (Beck 2000; Jeffries, Anderson et al. 2000; Wake 2001), the Dynamic System Development Method (DSDM Consortium 2009) and OpenUp (Eclipse Foundation 2009).

Agile processes are not without criticism (Boehm and Turner 2003; Stephens and Rosenberg 2003; Turk, France et al. 2005; Berard 2008). Among the problems frequently cited are: reliance on tacit knowledge, lack of scalability, the concept of simplicity and the assumption of the flat cost of change.

As illustrated by principle number 6 of the Agile Manifesto, Agile proponents privilege face-to-face meetings as coordination mechanism over documentation and plans. Critics point out that the focus on tacit knowledge makes projects that use Agile processes

dependent on experts and makes the transfer of knowledge outside the team difficult (McBreen 2002; Boehm 2002-a). As far as lack of scalability (Croker 2001), the Agile principles as originally stated were geared towards small, collocated teams. As this arrangement has proven to be a limitation, the Agile community has have to come up with concepts like the “scrum of scrum” (Sutherland 2005) and other hierarchical arrangements.

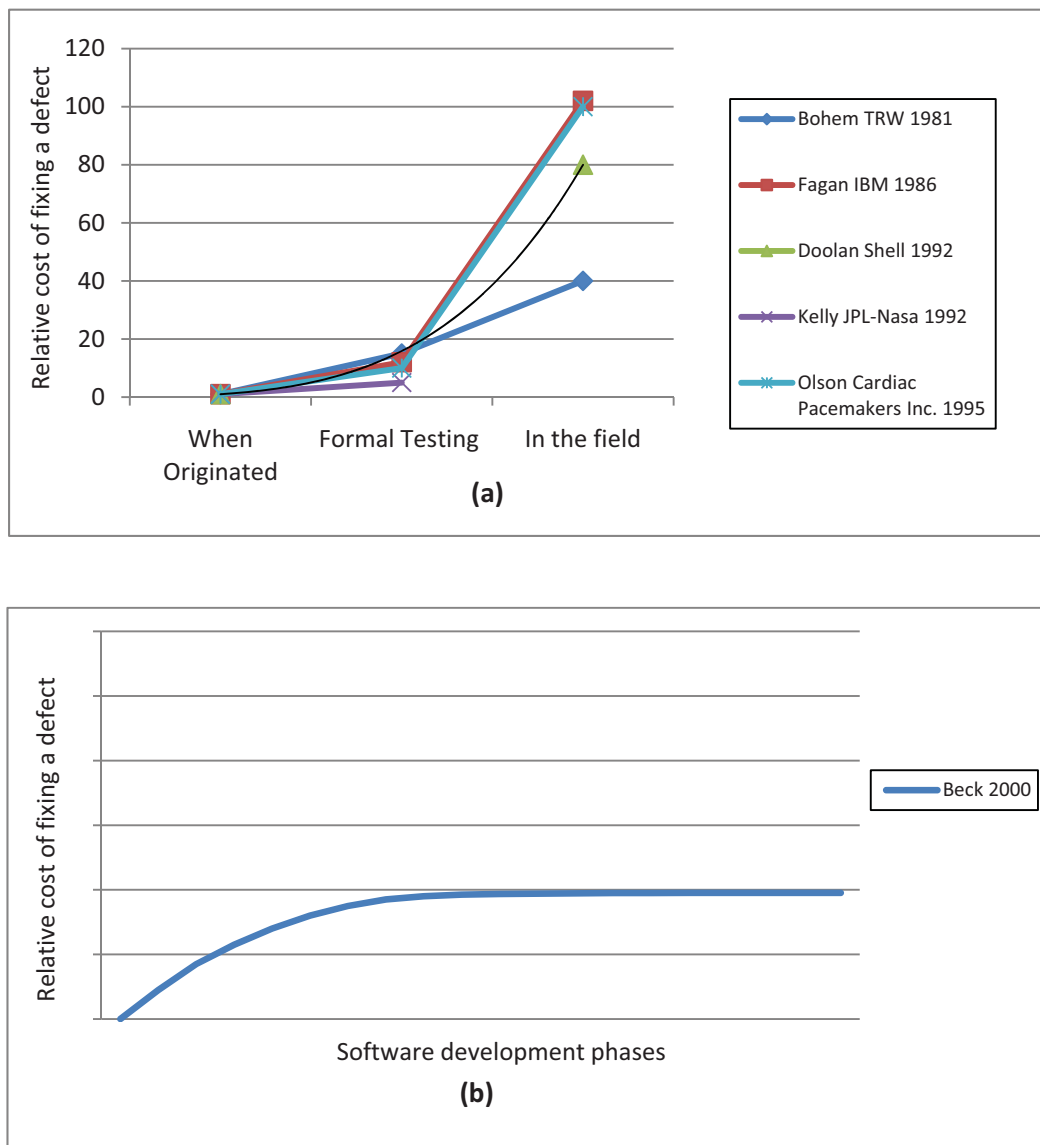
Simplicity, one of the four XP values (Beck 2000) and the emergence of design instead of a purposeful, system design activity, focuses developers on delivering just the functionality needed with no attempt to plan for features that might or might not be required in the future (Hunt 2005).

One of the most strongly criticized assumptions of the Agile movement is that of the “flat cost of change” (see Figure 7), expressed as follows:

“Under certain circumstances, the exponential rise in the cost of changing software over time can be flattened. If we can flatten the curve, old assumptions about the best way to develop software no longer hold” (Beck 2000)

It is on the basis of this statement that many of the Agile practices, such as refactoring and emergent design, find their economic justification for if the cost to correct the software increases exponentially as we move towards delivery, as documented by Boehm (Boehm 1981; Boehm 2002-b) and Doolan (Doolan 1992), then developing software according to these principles would be economically infeasible (Favaro 2003; Lattanze 2005).

The same aspects that could be viewed as disadvantages in certain contexts make Agile methodologies popular in sizeable segments of the software industry, like application composition, Web system development and system integration, which Boehm (Boehm, Clark et al. 1995) estimated as employing 1.4 million professionals in the US by 2005. The actual figure for the year 2008, according to the US Bureau of Labor Statistics, was 1.8 million, not including managers.

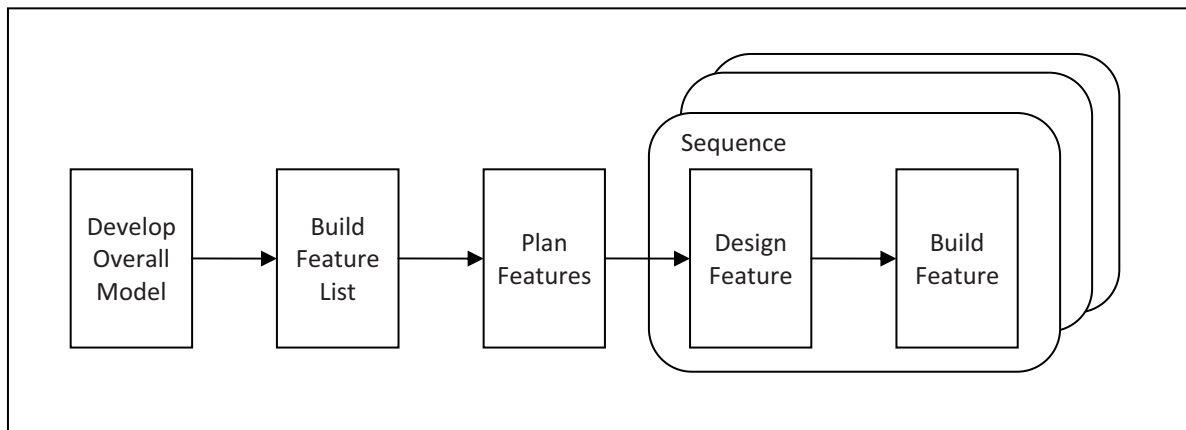


**Figure 7 Relative cost of fixing a defect**  
**(a) Traditional (Miranda 2003) and (b) as postulated by Beck (After Beck 2000).**

The next two sections introduce two Agile Development processes: Feature-Driven Development (FDD) and Scrum. The purpose of presenting these models is not to analyze them, but to provide the reader with concrete examples of representative processes

## 2.2 Feature-Driven Development

FDD is a short iteration process designed to deliver frequent, tangible, client-valued functionality called “features”, which form the basis for planning, reporting and tracking. The size of a feature is defined in terms of its end-to-end implementation effort, and is typically set at about two to three weeks. The FDD process begins by creating an object model of the software system to be built and a list of the features to be developed. Then, the features are implemented by adding the appropriate methods to the classes making up the object model. The overall FDD process is illustrated in Figure 8.



**Figure 8 Feature-Driven Development process  
(After Coad, Lefebvre et al. 1999).**

During the first activity of the process, the project members, under the guidance of an experienced component/object modeller (chief architect), build an object model of the software system. The purpose of this model is twofold: to define the scope of the system and its context, and to provide a “placeholder” in which the methods and attributes required by each feature will be hosted.

The second activity, building a features list, involves identifying the features, and then grouping and prioritizing them.

Based on the grouping and the dependencies between features defined during the “build a features list” activity, the project management team creates the development schedule,

decides on the number of resources, milestones, etc. The team also sets up the processes for monitoring and tracking.

For each feature identified, a design-by-feature and build-by-feature sequence is executed. There could be many of these sequences executing concurrently. In the design-by-feature activity, the chief programmer identifies the classes from the overall model likely to be involved in the feature, and contacts the corresponding class owners. This feature team, which includes the chief programmer and the class owners, works out a detailed sequence diagram, and the class owners write class interfaces. Finally, the team conducts a design inspection. Each class owner then builds his methods for the feature, extends the class-based test cases and performs class-level (unit) testing. The feature team inspects the code. Once the code is successfully implemented and inspected, the class owner checks the class(es) into the configuration management system. When all the classes for this feature are checked in, the chief programmer promotes the code to the build process.

In FDD, work is allocated using a matrix approach (see Table 2), where one of the dimensions lists all the features and the other the owners of the intervening classes.

Table 2 Work Allocation in Feature-Driven Development Projects

Class Feature		Class 1	Class 2	...	Class n
	Owner	John	Ellen		David
Feature 1	Ellen	x	x		
Feature 2	Ellen		x		x
Feature 3	David	x	x		
.....					
Feature n	John		x		x
An x at the intersection of a column and a row indicates that the class participates in the implementation of the feature					

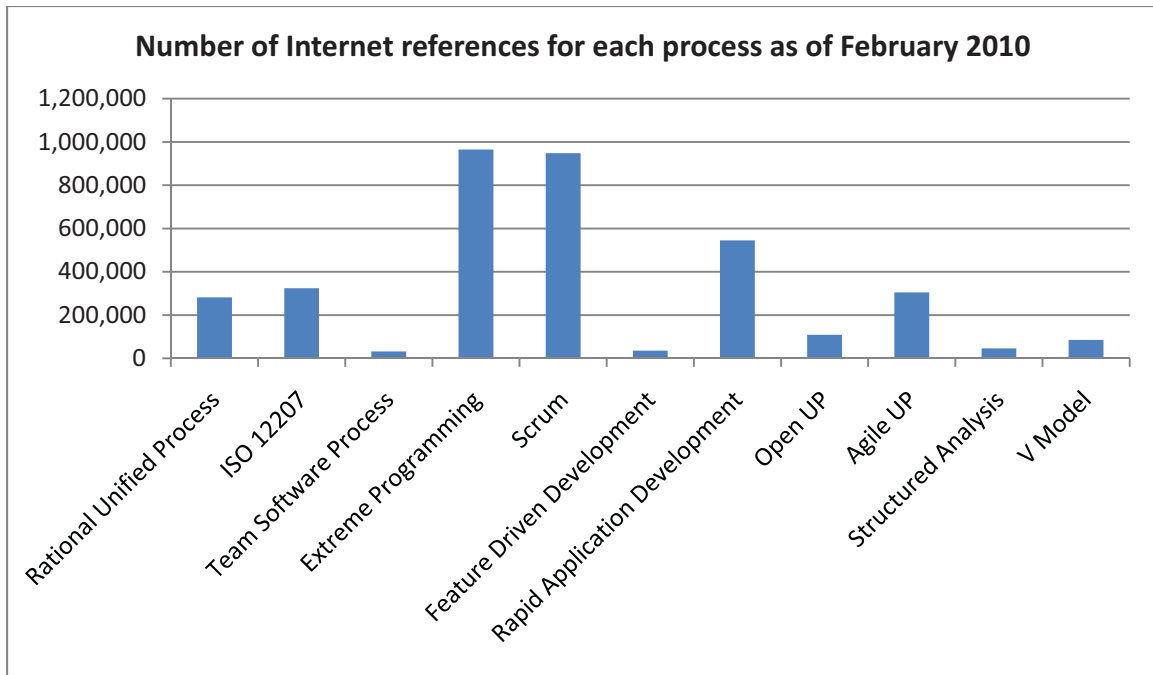


## 2.3 Scrum

Scrum (see Figure 9) is a widely adopted management process for software development projects initially proposed by Ken Schwaber and Jeff Sutherland (Schwaber and Beedle 2001). Its main characteristics are the following:

- Self-organizing teams, with only three roles specified;
- Requirements captured as items in a “product backlog” list;
- Product progression in a series of month-long “sprints”;
- Requirements converted into tasks and documented in the sprint backlog at the beginning of each sprint;
- Each sprint ends with a potentially shippable product, and the work performed is reviewed with the product owner;
- Software effectively transferred to the user (shipped, delivered, put into production, etc.) is called a release, which might comprise work performed in more than one sprint;
- No specific engineering practices prescribed;
- Daily team meetings, the Scrum meeting, to discuss what was done, what is next and what obstacles have been encountered.

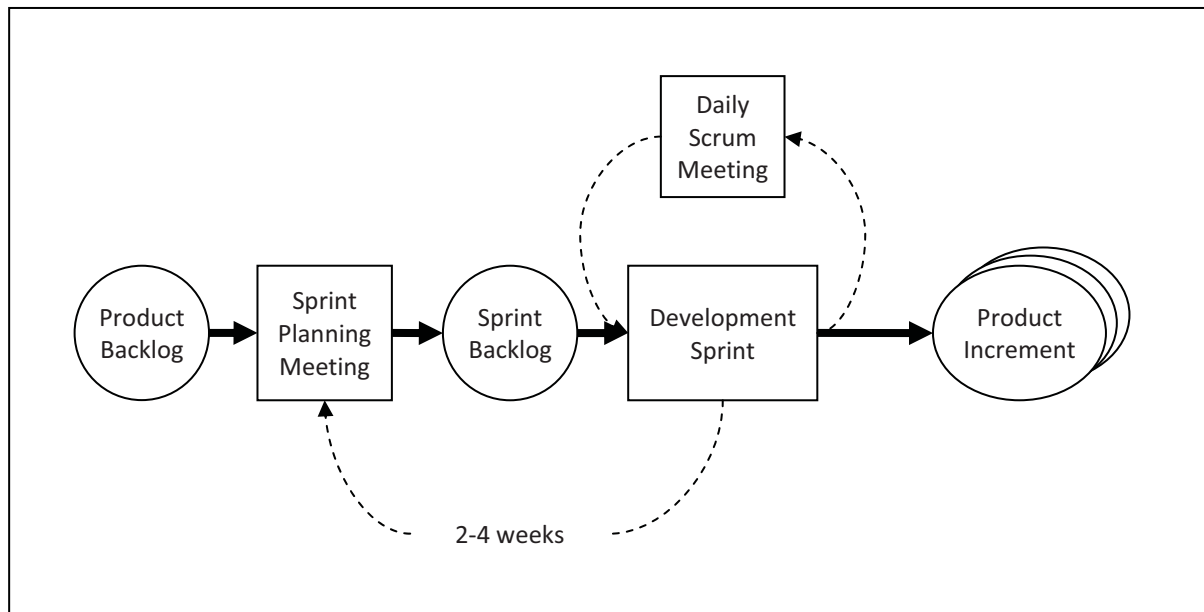
Work is organized around two time boxes that are executed repeatedly (see Figure 10). The larger iteration is called a Sprint, and lasts two to four weeks. The smaller is called a Scrum and repeats daily. The Sprint iteration is used as the basis for planning and user validation of progress, while the Scrum is used to coordinate work by members of the team. Scrum roles, artifacts and management practices are listed in Appendix A.



**Figure 9 Diffusion of various software development processes, as measured by the number of references returned by a Google search.**

Scrum is a minimal process, in the sense that is difficult to imagine how a team could be successful at developing a software system without having a “to do” list (the product and the sprint backlogs), some kind of coordination (the Sprint and the Scrum meetings) and some governance scheme (the product owner and the scrum master).

This chapter has provided a brief introduction to Agile development and two representative methods. The next chapter will present an overview of the published articles, their contribution and position with respect to the Guide to the Software Engineering Body of Knowledge (Abran, Moore et al. 2004).



**Figure 10 The Scrum process**  
(After Cohn 2009).

## **CHAPTER 3**

### **MAJOR THEMES**

Three research themes pervade this thesis: the estimation of software size for the purpose of planning a project, the calculation and administration of contingency funds and the monitoring of development activities. The first theme is addressed in the article “*Sizing User Stories Using Paired Comparisons*” (Annex I), which describes a method to support judgement-based effort estimation. The second theme is the subject of the article “*Protecting Software Development Projects Against Underestimation*” (Annex II), which proposes a new method of calculating contingency funds that takes into consideration the human and organizational considerations that affect decision-making in a project facing overruns. The third theme, the monitoring of development activities, is addressed in the article “*Agile Monitoring Using the Line of Balance*” (Annex III), which proposes a monitoring method that is almost a by-product of team activities and operations over a version control system, and not a heavy reporting superstructure imposed on the developers.

This chapter provides a brief introduction to each of the publications, highlighting their contributions to the software project management discipline. They are then analyzed with respect to the Guide to the Software Engineering Body of Knowledge to demonstrate that they form a cohesive whole, as they all relate directly to software engineering management in general, and to Agile projects in particular.

#### **3.1        Sizing User Stories Using Paired Comparisons**

To estimate the amount of work required in a project, Agile development relies on the judgement of the participating developers combined through a number of mechanisms, which include Ad Hoc consultations, Delphi-like techniques and statistical groups (Molokken, Haugen et al. 2008). The Paired Comparison method for software estimation (Bozoki 1993; Miranda 1999; Miranda 2000; Shepperd and Cartwright 2001; Miranda 2001a; Hihn and Lum 2004) is designed to support expert estimation by comparing the relative size of the features that drive the planning of the development project, which improves both the

accuracy and the precision of estimates (Sasao 2009) compared to Ad Hoc estimates and to Planning Poker (Grenning 2002; Cohn 2006), a popular variation of the wideband Delphi method described by Boehm (Boehm 1981). The major drawback of the Paired Comparison method is the large number of judgements required to arrive to an accurate estimate.

The method proposed in this thesis (Miranda, Bourque et al. 2009) and described in detail in Annex I, uses incomplete cyclic designs (Spence and Domoney 1974; Spence 1982; Burton 2003) to select which comparisons to make. The contribution of the redesigned method rests in the reduction of the number of comparisons required, up to 35% relative to the full factorial model, without a corresponding loss in the reliability of the estimates. This redesign allows the method to be applied to larger sets, or, conversely, to estimate a set of a given size with less effort.

Software estimation using Paired Comparisons is especially well suited to the early stages of a development project, when the knowledge available to project team members is mostly qualitative. The relevance of the method has been established by Jørgensen and Shepperd's writing (Jorgensen and Shepperd 2007), calling for the development of techniques that support rather than replace expert judgment, the prevailing practice in industry.

### **3.2 Protecting Software Development Projects against Underestimation**

Agile projects are not immune to overruns, delays and bad business decisions based on poor estimates. The promise of trying their best and setting the scope, budget and/or schedule as the project progresses (Sutherland and Schwaber 2007) would not be enough for stakeholders requiring a fixed scope and schedule to formulate their own plans and budgets. For this, it is necessary to calculate contingencies to bring the risk associated with any project to levels acceptable to them.

In "Protecting Software Development Projects Against Underestimation" (Miranda and Abran 2008), an algorithm is proposed to do this. The contribution of the article, included as

Annex II of this thesis, is threefold. First, the author postulates that a realistic calculation of contingency funds should be based on the cost of keeping the project on schedule, and not on what it would have cost had the work been planned from the beginning. Second, the model uses a new communication model derived by the author based on his working experience (Miranda 2001b), and which he later found corroborated by the data collected by Allen at MIT (Allen 1984). Third, it demonstrates the need to administer the contingency funds above the project level to take advantage of cost savings achieved by projects which are under cost, so that they are passed on to elements requiring additional resources.

As part of the analysis, the author also qualifies Brook's old adage that "*adding a person to a late project makes it later*" (Brooks 1995), by demonstrating that, while adding a person to a late project is more expensive than adding it at the beginning of the project, only under very extreme circumstances, e.g. gross underestimations acknowledged late in the development life cycle, does the addition of the extra resource result in process losses so severe that the project is delayed.

### 3.3 Agile Monitoring Using the Line of Balance

Progress monitoring and reporting is a basic function during the execution of any project. Progress needs to be monitored so that potential problems can be avoided before they materialize. Besides being required to steer the project, timely and accurate reporting is essential to keep stakeholder support and funds flowing. Agile projects are no different.

In the article, “Agile Monitoring Using the Line of Balance” (Miranda and Bourque 2009) (see Annex III), the authors propose the use of line-of-balance calculations (Office of Naval Material 1962; Al Sarraj 1990; Arditi, Tokdemir et al. 2002; Harroff 2008) to track progress and to balance resources in Agile projects.

The contribution of this article resides in the novel application of an old method to software development. In keeping with the idea of using actual data for planning, the authors replaced the target or standardized times used in the original method with data captured by a version control system.

The line-of-balance (LOB) technique was devised by the members of a group headed by George E. Fouch during the 1940s to monitor production at the Goodyear Tire & Rubber Company which the author has successfully used to monitor the progress of fixing the trouble reports while working at Ericsson . The LOB method comprises the designation of a number of control points, together with their lead times, and a target plan displaying the cumulative production schedule as planned by the project manager. The control points’ leadtimes are used to calculate how many units should have passed through each of them as of a given date vs. how many actually passed. By comparing these two numbers, it is possible to assess the project progress and any unbalance that might exist in the allocation of resources. The latter is an important advantage of the proposed method over other tracking approaches used in Agile projects, since unbalanced operations result both in inefficient utilization of resources and in throughput losses.

### **3.4 The Articles in Context**

The underlying theme of the three articles is the improvement of the management of software development projects in general and Agile projects in particular. To better understand this connection, the articles are analyzed here against the knowledge taxonomy proposed in the Guide to the Software Engineering Body of Knowledge (Abran, Moore et al. 2004).

#### **3.4.1 The Guide to the Software Engineering Body of Knowledge**

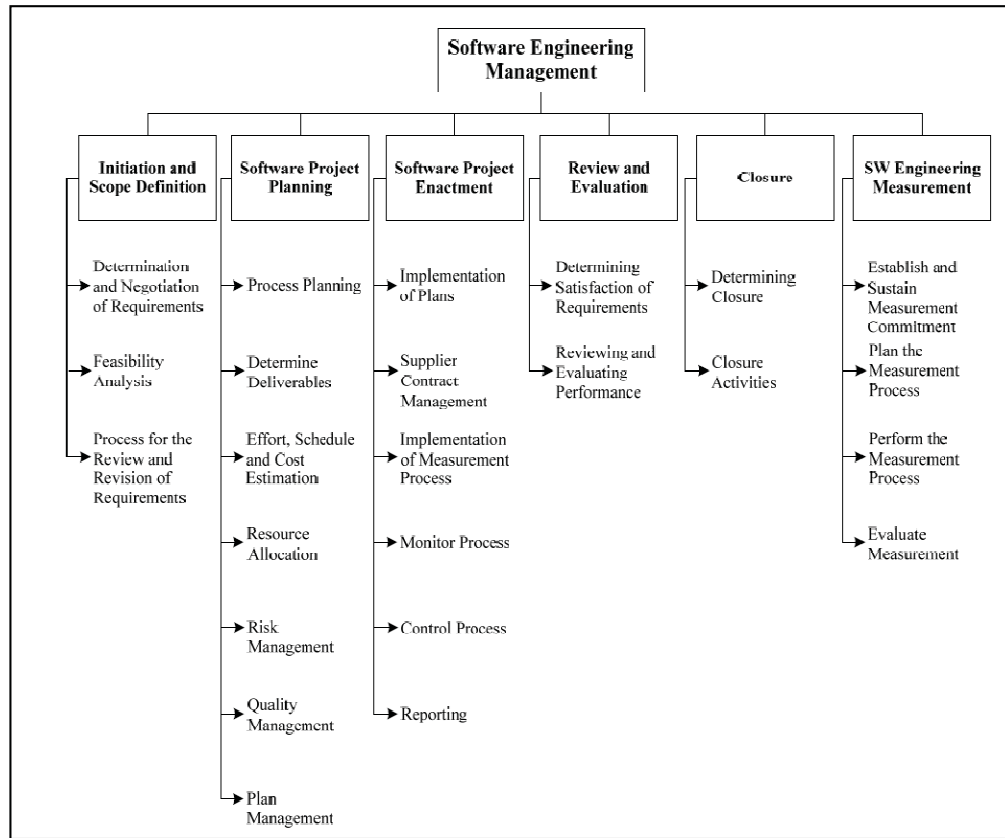
The Guide to the Software Engineering Body of Knowledge (SWEBOK) is a comprehensive collection of generally accepted knowledge within the academic and industrial software engineering communities. The phrase “generally accepted” is taken to have the same meaning as the phrase “generally recognized” in the Project Management Body of Knowledge (Project Management Institute 2004), which states: “Generally recognized” means that the knowledge and practices described are applicable to most projects most of the time, and that there is widespread consensus about their value and usefulness”

The content of the SWEBOK is organized into the following 10 knowledge areas:

- Software requirements,
- Software design,
- Software construction,
- Software testing,
- Software maintenance,
- Software configuration management,
- Software engineering management,
- Software engineering process,
- Software engineering tools and methods,
- Software quality,

Each of the areas is further decomposed into sub-areas, topics and subtopics. Figure 11 shows the structure of the Software Engineering Management Knowledge Area, into which all the topics addressed in this thesis fall.





**Figure 11 Software Engineering Management Knowledge Area topics (Abran, Moore et al. 2004). Copyright © 2004 by The Institute of Electrical and Electronics Engineers, Inc. All rights reserved.**

### 3.4.2 Traceability between the Publications and the SWEBOK

Table 3 traces each of the publications to the corresponding topic or subtopic in the SWEBOK. The presence of two marks on a given row of the traceability table, e.g. the Effort, Schedule and Cost Estimation row, which contains two x's, one corresponding to the Sizing User Stories Using Paired Comparisons publication and the other to the Protecting Software Development Projects Against Underestimation publication, denotes a connection between the corresponding themes addressed by them, e.g. in order to estimate the amount of contingency necessary for a budget, I first need to know the budget.

The correspondence between topics and subtopics in the SWEBOK and each article was established by comparing their respective content. The corresponding definition for the SWEBOK topics and subtopics is provided in Appendix B for the sake of completeness.

Table 3 SWEBOK – Publications Traceability Matrix

Knowledge Areas	SWEBOK's Topics	Sizing User Stories Using Paired Comparisons	Protecting Software Development Projects Against Underestimation	Agile Monitoring Using the Line Of Balance
Initiation and Scope Definition	Determination and Negotiation of Requirements			
	Feasibility Analysis (Technical, Operational, Financial, Social)	X		
	Process for the Review and Revision of Requirements			
Software Project Planning	Process Planning			
	Determine Deliverables			
	Effort, Schedule and Cost Estimation	X	X	
	Resource Allocation			
	Risk Management		X	
	Quality Management			
	Plan Management			
Software Project Enactment	Implementation of Plans			
	Supplier Contract Management			
	Implementation of Measurement Process			
	Monitor Process			X
	Control Process		X	X
	Reporting			X
Review and Evaluation	Determining Satisfaction of Requirements			
	Reviewing and Evaluating Performance			
Closure	Determining Closure			
	Closure Activities			
Software Engineering Measurement	Establish and Sustain Measurement Commitment			
	Plan the Measurement Process			
	Perform the Measurement Process			
	Evaluate Measurement			

## CONCLUSION

The publications that make up the core of this thesis present three novel methods that can be employed by practitioners in the Agile and other communities to plan and monitor their software development projects. Their relevance and correctness has been established by a peer review process implemented by three different journals: Information and Software Technology, the Project Management Journal and the Journal of Systems and Software.

The main results of this research are:

- The development of a scalable and reliable method to support the estimation of Agile development projects. An empirical verification of the method was conducted using two datasets, showing that the proposed method produces good estimates, even when the number of comparisons is reduced by half those required by the original formulation of the Paired Comparison method.
- The development of a model to calculate contingency funds that takes into account the human and organizational considerations that govern decision making in real projects and the verification of the need to administer these funds above the project level. This last point was verified by studying a number of different scenarios using Monte Carlo simulations.
- The development of a technique to effectively and efficiently track progress on Agile projects. The method is effective and efficient in the sense that it not only summarizes information currently dispersed in a number of project artifacts, but also provides new insights into balancing the workload and combines this with the overall project plan.

While the methods proposed constitute a contribution to the software engineering body of knowledge, there is still research work to be carried out.

A current limitation of the Paired Comparison method is that it includes, in one judgement, at least two dimensions that are correlated with the effort to program a certain feature: its length

and its complexity. In other words, a given piece of software can be longer, i.e. more lines of code than another, but its logic can be simpler, and it is up to the estimator to balance them when comparing one entity to another.

Any model is a compromise between fidelity and analytical tractability. The geometric model used to calculate contingency funds is no exception. For example, the original distribution of effort is assumed to be uniform when for some type of projects a more realistic representation would be based on a back loaded curve. Similarly, the probability distribution of the time to acknowledge the underestimation is a simple arithmetic progression. Other possibilities that ought to be researched include the use of conditional probabilities to model the effect of the underestimation, e.g., larger underestimations will be easier to notice than smaller ones and right skewed distribution that will emphasize the effect of procrastination in deciding whether to re-plan a project or not.

Implicit in the current version of the LOB method is the assumption that the production rate remains constant throughout the project. However, years of study of work in industrial plants has demonstrated the existence of learning curves that result in productivity increases as workers become more proficient in the production process. It is not too radical a speculation that, as the development team becomes more familiar with itself and with the product under development, productivity will go up as well. Employing a straight line, as in the current method, instead of a curve reflecting learning will result in an underestimation of the progress achieved so far.

In addition to the two points mentioned above, future research work includes the substitution and evaluation of the current imputation method and the revision of the calculations concerning the propagation of errors associated with individual judgements.

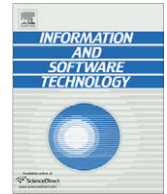
The current version of the Paired Comparison method utilizes the mean value of the row of judgements to assign values to the comparison skipped by virtue of the design. The rationale for the selection of this method is that, since the comparisons included in a particular ICD

share no particular order, the values missing do so at random. This choice however affects the value of the inconsistency index, and the question is whether an imputation technique that exploits the  $a_{ij} \times a_{jk} = a_{ik}$  relationship of a perfectly consistent matrix could be a better one.

With regard to the propagation of errors to individual judgements, the current version of the method assumes that every individual judgement contributes in the same proportion to the total error represented by the inconsistency index. If this were not the case, i.e. judgements between entities further apart yielded larger, or smaller, errors, then the current method would underestimate, or overestimate, the prediction intervals.

## **ANNEX I**

### **Sizing User Stories Using Paired Comparisons**



## Sizing user stories using paired comparisons

Eduardo Miranda <sup>a,\*</sup>, Pierre Bourque <sup>b</sup>, Alain Abran <sup>b</sup>

<sup>a</sup> Institute for Software Research, Carnegie-Mellon University, Pittsburgh, Pennsylvania, PA, United States

<sup>b</sup> École de Technologie Supérieure, Université du Québec, Canada

### ARTICLE INFO

#### Article history:

Received 16 February 2009

Received in revised form 5 April 2009

Accepted 10 April 2009

Available online 18 April 2009

#### Keywords:

Agile estimation

Triangulation

Estimating by analogy

Paired comparisons

Software sizing

Expert estimation

Expert judgment

Project planning

Incomplete cyclic designs

### ABSTRACT

Agile estimation approaches usually start by sizing the user stories to be developed by comparing them to one another. Various techniques, with varying degrees of formality, are used to perform the comparisons – plain contrasts, triangulation, planning poker, and voting. This article proposes the use of a modified paired comparison method in which a reduced number of comparisons is selected according to an incomplete cyclic design. Using two sets of data, the authors show that the proposed method produces good estimates, even when the number of comparisons is reduced by half those required by the original formulation of the method.

© 2009 Elsevier B.V. All rights reserved.

### 1. Introduction

Agile estimation approaches typically comprise three steps: (1) comparison of the user stories to be developed to one another for the purpose of establishing their relative size; (2) conversion of the size estimates to lead times using an assumed team productivity; and (3) re-estimation of the project lead times using the team's actual productivity, once this becomes known after two or three iterations.

User story comparisons take the following form: “This story is like that story, so its size must be roughly the same,” or “This story is a little bit bigger than that story which was estimated at 4, so its size should be around 5.” The numbers 4 and 5 in the previous sentence are called “story points”, which are numbers in ratio scale purportedly proportional to the effort it would take to develop each story based on its perceived size and complexity [1]. A 6-point user story is expected to require about twice as much effort as a 3-point user story. The degree of structure in the comparison process ranges from the ad hoc comparison of any two user stories, to triangulation – the comparison of a user story with two others, to a number of Delphi [2] like techniques such as the planning poker [3]. To avoid wasting time discussing insignificant differences between user stories, the use of a Fibonacci or power series is sometimes recommended, such as if the difference between two user

stories is not as large as a following term in the series, the two user stories are assumed to be of the same size [4].

The project lead time is calculated using the concept of *velocity*, which is a proxy for the productivity of the team. At first, velocity is estimated or taken from a previous project, but, as work progresses, it is measured by tallying the number of story points completed during the counting period. Velocity is measured in story points per iteration, or story points per month. As an example, if the current team velocity is 30 story points per month, it will take the team 2 months to deliver 60 story points-worth of user stories.

As will be shown later, comparing one user story to another, or to two others, is not good enough to produce reliable estimates. The first reaction to this is to increase the number of comparisons, but this creates some problems of its own. As even the most devoted estimator gets tired after making a large number of comparisons, the question of how many comparisons to make becomes really important, as does the problem of dealing with the inconsistencies inherent to the judging process.

To address these problems, we propose the use of incomplete cyclic designs to identify which user stories to compare with which to reach a desired accuracy, and the use of the paired comparison method [5–7] to deal with judgment inconsistencies.

The rest of the paper is organized as follows: Section 2 formalizes the triangulation concept, Section 3 explains the basic paired comparison method, Section 4 presents the modified process using incomplete cyclic designs, Section 5 discusses the accuracy and

\* Corresponding author.

E-mail address: [mirandae@andrew.cmu.edu](mailto:mirandae@andrew.cmu.edu) (E. Miranda).

precision of the resulting estimates, and Section 6 provides a summary of the article.

## 2. Agile estimation and triangulation

Triangulation is defined in the Agile literature as the process of establishing the size of a user story relative to two other user stories with the purpose of increasing the reliability<sup>1</sup> of the estimate [3]. When using triangulation, the comparisons sound something like this: “I’m giving user story B 2 points, because it feels like its implementation will take somewhat more effort than user story A, which I already rated at 1 story point, and somewhat less effort than user story C, which I rated as a 4-point story.” Despite its intuitive appeal, triangulation is not as simple as the sentence above makes it appear. First, there is the problem of consistency, which can be mathematically expressed as:

$$a_{ij} \times a_{jk} = a_{ik} \quad \forall i, j, k \in n \quad (1)$$

Eq. (1) reads as follows: if user story<sub>i</sub> is  $a_{ij}$  times bigger<sup>2</sup> than user story<sub>j</sub>, and user story<sub>j</sub> is  $a_{jk}$  times bigger than user story<sub>k</sub>, then user story<sub>i</sub> must be  $a_{ij} \times a_{jk}$  times bigger than user story<sub>k</sub>. This is important, because lack of consistency among triangulations leads to inaccurate estimates.

Second, which two user stories should you choose as reference points? Does the choice affect the result?

The triangulation process can be visualized by arranging the user stories in a circular pattern and linking those being compared (see Fig. 1). Given  $n$  user stories to be estimated, there are  $n(n-1)(n-2)/2$  possible configurations or designs which can be evaluated, but not all are equally good. A good design must have two properties: balance and connectedness [8–10]. A design is considered balanced when every user story appears in as many comparisons as any other user story. This ensures that one user story does not overly influence the estimation, while others are under-represented. Connectedness implies that any user story is compared, directly or indirectly, to every other user story. An unconnected graph is undesirable, because the size of some user stories relative to others would be completely indeterminate. Fig. 1b illustrates the problem: the user stories in the lower subset are never compared against those in the upper subset, so each subset could be accurately sized in itself but completely offset with respect to the other.

The number of times a user story appears in a comparison is called the replication factor ( $r$ ) of the design. In all the designs shown in Fig. 1,  $r$  is 2.

Balance and connectedness are necessary, but not sufficient conditions for a good estimation. As shown by Burton [8], a low  $r$ , such as that used in the triangulation approach ( $r = 2$ ) is very sensitive to errors in judgment, and thus tends to produce unreliable results. In his experiments, Burton found that the correlation ( $\rho$ ) between the actual and the estimated values using triangulation ranged from a low of 0.46 to a high of 0.92, with a mean value of 0.79. Similar variability was found by the authors using two sets of data, this is discussed later.

## 3. Paired comparison method basics

### 3.1. Overview

The idea behind the paired comparison method is to estimate the size of  $n$  user stories by asking one or more developers to judge

their relative largeness rather than to provide absolute size values. After this is done, one of the  $n$  user stories is assigned an arbitrary number of story points. Using this story as reference, the sizes in story points, of all the other user stories are calculated. The process is called Full Factorial Pairwise Comparison because it compares all user stories (factors) against one another, see Fig. 2.

Although the selection of the user story to be used as reference and the allocation of story points to it is arbitrary to a certain point,<sup>3</sup> a consistent selection and allocation, i.e. two comparable user stories are not allocated 4 story points in one project and 10 in other, is useful for the developers to develop an intuition or sense for the effort required in the realization of a user story with so many story points.

It is also possible to use the method to estimate the effort required by each user story instead of their story points. In this case, a user story whose development effort, from either a previous project or a spike,<sup>4</sup> is known will be brought in as reference story. For a more detailed description of the method, refer to [5,6].

In the rest of the document we will work with story points to remain true to the title of the essay but all the same concepts apply to the calculations using effort.

### 3.2. The pairwise comparison of user stories

Developers start the process by judging the relative size ( $a_{ij}$ ) of each user story against every other user story, and recording these values in a matrix called the judgment matrix (2).

$$A^{n \times n} = \begin{cases} a_{ij} = \frac{sp_i}{sp_j} & \text{How much bigger(smaller)user story}_i \\ & \text{is with respect to user story}_j \\ a_{ii} = 1 & \text{Every user story has the same size as itself} \\ a_{ji} = \frac{1}{a_{ij}} & \text{If user story}_i \text{ is } a_{ij} \text{ times bigger (smaller)} \\ & \text{than user story}_j, \text{ then user story}_j \\ & \text{is } 1/a_{ij} \text{ times smaller(bigger)than user story}_i \end{cases} \quad (2)$$

$sp_i$  and  $sp_j$  are the as yet unknown numbers of story points for user story<sub>i</sub> and user story<sub>j</sub> to be derived from the  $a_{ij}$  judgments. Note that only the comparisons corresponding to the upper diagonal matrix have to be made, since the  $a_{ji}$  are the reciprocals of the  $a_{ij}$ .

### 3.3. Calculating the size, the Inconsistency Index, and the standard deviation

Once all the  $a_{ij}$  judgments have been recorded in the judgment matrix, the mean relative size ( $mrs_i$ ) of user story<sub>i</sub> is calculated as the geometric mean [11,12] of the  $i$ th row (3) of the judgment matrix. The size in story points of each user story is then computed by multiplying its  $mrs_i$  by the normalized size of the reference user story (4). For a more detailed description of the method, refer to [5,6].

$$mrs_i = \left( \prod_{j=1}^n a_{ij} \right)^{\frac{1}{n}} \quad (3)$$

$$sp_i = \frac{sp_{reference}}{mrs_{reference}} \times mrs_i \quad (4)$$

As inconsistencies are inherent to the judgment process, Crawford and Williams [12] and Aguaron and Moreno-Jimenez [13] suggest

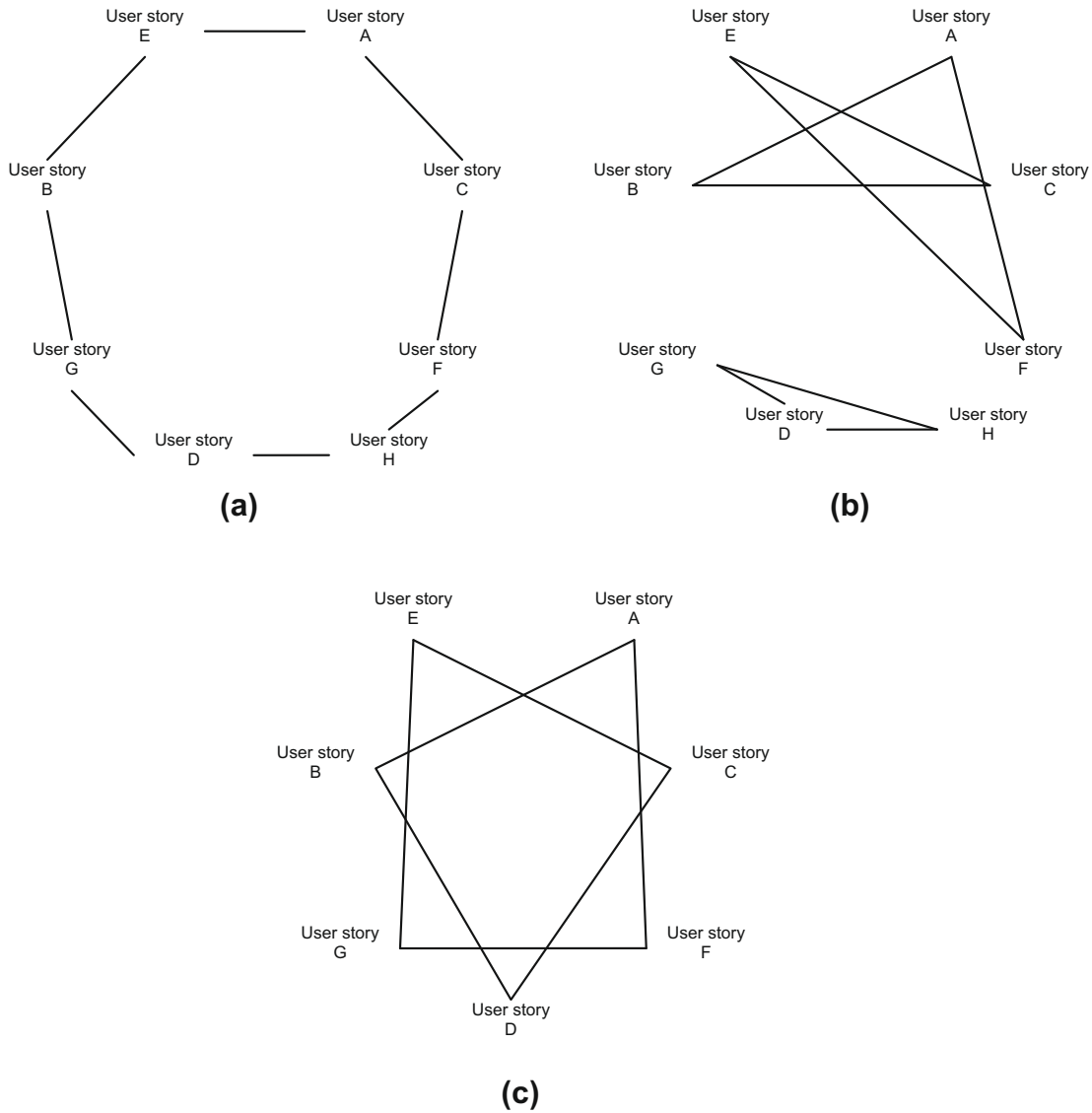
<sup>1</sup> A reliable sizing method will yield estimates that are accurate, that is, close to their true value, and precise, that is estimates must be consistent across repeated observations in the same circumstances.

<sup>2</sup> The comparison can go both ways, i.e. replacing bigger for smaller.

<sup>3</sup> The number zero must be reserved for “stories” with not content to preserve the properties of a ratio scale.

<sup>4</sup> In the Agile terminology a spike is an experiment that is performed to learn something. In this case the spike would consist on developing a user story tracking how much effort it required.





**Fig. 1.** Three triangulation designs out of the  $n(n-1)(n-2)/2$  possible ones. Note 1: 'a' and 'c' are good designs, but 'b' is not, as it consists of two disjoint subgraphs.

the use of (5) as an unbiased estimator of the variance of the inconsistencies of the judgment matrix  $A^{n \times n}$ . The larger the inconsistencies between comparisons, the larger the variance will be. The square root of (5) is called the Inconsistency Index (6) of the judgment matrix.

$$\sigma_A^2 = \frac{\sum_{i < j}^n \left( \ln a_{ij} - \ln \frac{mrs_i}{mrs_j} \right)^2}{\frac{n(n-1)}{2} - (n-1)} \quad (5)$$

$$InconsistencyIndex = \sqrt{\sigma_A^2} = \sqrt{\frac{2 \sum_{i < j}^n \left( \ln a_{ij} - \ln \frac{mrs_i}{mrs_j} \right)^2}{(n-1)(n-2)}} \quad (6)$$

While the Inconsistency Index gives an overall idea of the quality of the judgments, it is a quantity that is difficult to interpret. A much better alternative is to present the estimator with a range estimate – an interval within which the estimate will likely fall for a given degree of inconsistency.

To calculate the extremes of the interval, we start by assuming that each user story contributes equally to  $\sigma_A^2$ . This assumption allows us to write Eq. (7), where  $\sigma_A^2$  is shown to result from the sum

of  $n$  individual inconsistencies  $\sigma_i^2$  contributed by each user story.<sup>5</sup> The standard deviation of the size of each user story could then be calculated as the product of its estimated size and its individual inconsistency (8). The range estimate is given by (9).

$$\sigma_A^2 = \sum_{i=1}^{n-1} \sigma_i^2 = n\sigma_i^2$$

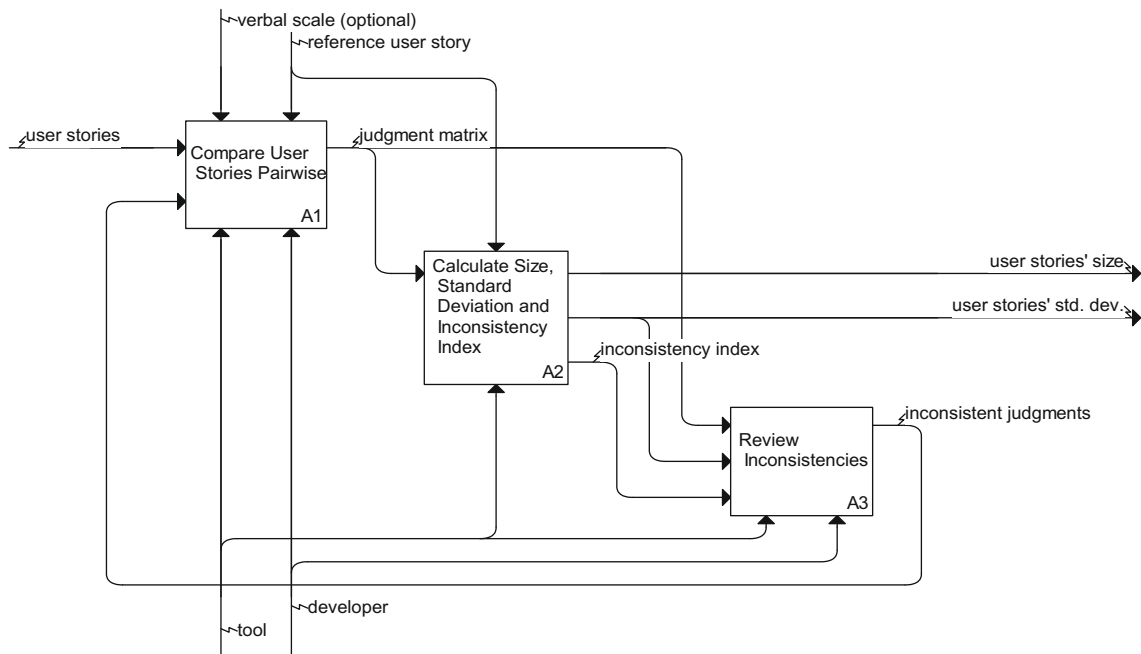
$$\text{therefore } \sigma_i = \sqrt{\frac{\sigma_A^2}{n}} \times \frac{InconsistencyIndex}{\sqrt{n}} \quad (7)$$

$$\sigma_{sp_i} = sp_i \times \frac{InconsistencyIndex}{\sqrt{n}} \quad (8)$$

$$RangeEstimate = [sp_i - \sigma_i, sp_i + \sigma_i] \quad (9)$$

Other approaches to calculating the standard deviation of the size exist. Hihn and Lum [14] proposes the use of a triangular distribu-

<sup>5</sup> If the reference story is brought in from another project or from a spike the denominator in Eqs. (7) and (8) needs to be replaced by  $n-1$  instead of  $n$  to account for inclusion of the known parameter.



**Fig. 2.** In the Full Factorial paired comparison process. Each user story is compared to every other user story. Note 1: The optional verbal scale allows the user stories to be compared using an ordinal scale by labeling the comparison of two user stories with adjectives such as “equal”, “a little bigger”, “much bigger”, etc.

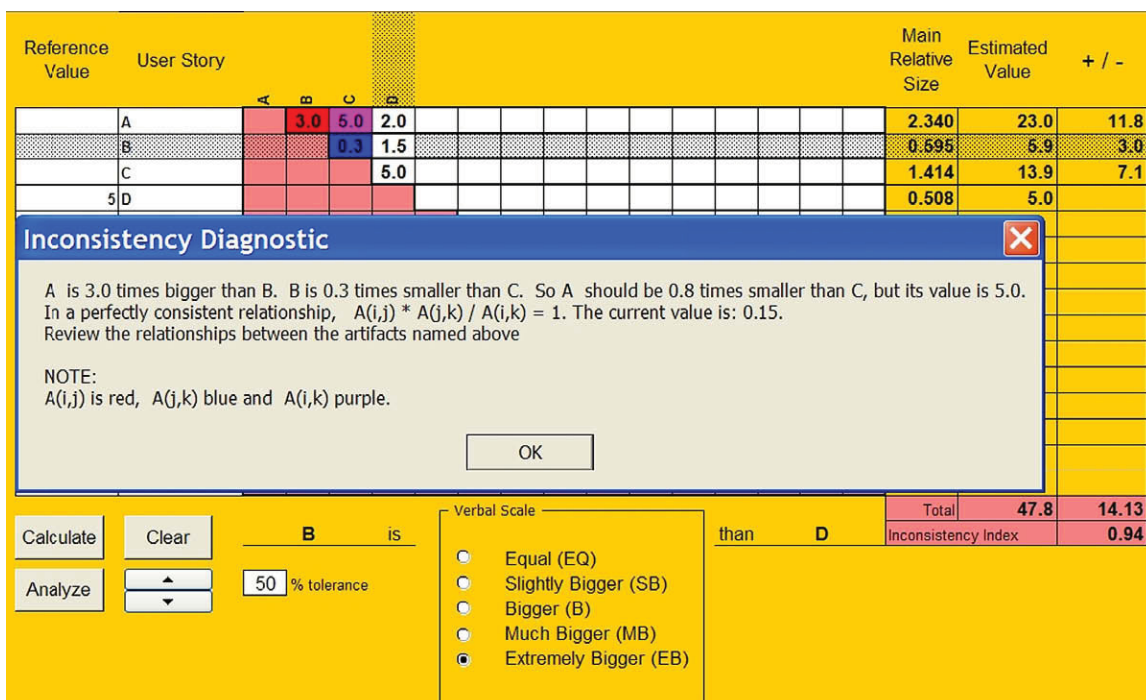
tion, where the developer or estimator judges the relation between two user stories in terms of the best case, the most likely case, and the worst-case scenarios, but, given the rather large number of user stories included in a typical project, we found this to be too taxing.

### 3.4. Reviewing inconsistencies

At this point, the estimator will use the Inconsistency Index or the estimates' range as a guide to decide whether or not these

are good enough and stop, or revise all or some of his judgments with the objective of reducing the inconsistencies.

By simulating a large number of judgment matrices and comparing them to perfectly consistent ones, Aguaron and Moreno-Jimenez [13] determined that, for sets with four or more data points, an Inconsistency Index less than or equal to 0.35 would produce satisfactory results in most cases. To give the reader an idea of its meaning, an Inconsistency Index of 0.35 with 15 user stories being estimated would, under the assumption that all user



**Fig. 3.** Tool interface for detecting the most inconsistent judgments. Note 1: Each time the “Analyze” button is pressed, a new triad is displayed. Note 2: The “spinner” is used to specify the amount over which a given triad is considered inconsistent. Note 3: The values in the matrix and the estimated value are rounded to the nearest digit for display purposes.

stories contribute equally to it, result in a size range of  $\pm 9\%$  for each story. If fewer user stories are compared, the size range will be wider. If more user stories are compared, the interval will be narrower.

### 3.5. A numerical example

Suppose we wanted to estimate the story points of four user stories called A, B, C and D. If we judge the size of A to be three times that of B, five times that of C, and twice that of D, and then we assess B to be roughly a quarter of C and one-and-a-half times D, and C as being five times bigger than D, the resulting matrix is:

	A	B	C	D
A	1	3.0	5.0	2.0
$A^{4 \times 4} = B$	.33	1	.25	1.5
C	0.2	4.0	1	5.0
D	.50	.67	.20	1

Only the relative size of the upper diagonal elements of the matrix needs to be judged, as all the other values can be derived using the definitions in (2). Applying Eq. (3), the mean relative size of each user story is:

$$mrs_i = \begin{cases} 2.34 \\ 0.59 \\ 1.41 \\ 0.50 \end{cases}$$

By designating D as the reference user story and assigning it 5 story points we anchor the scale above and are able to calculate the size of the other user stories using Eq. (4)

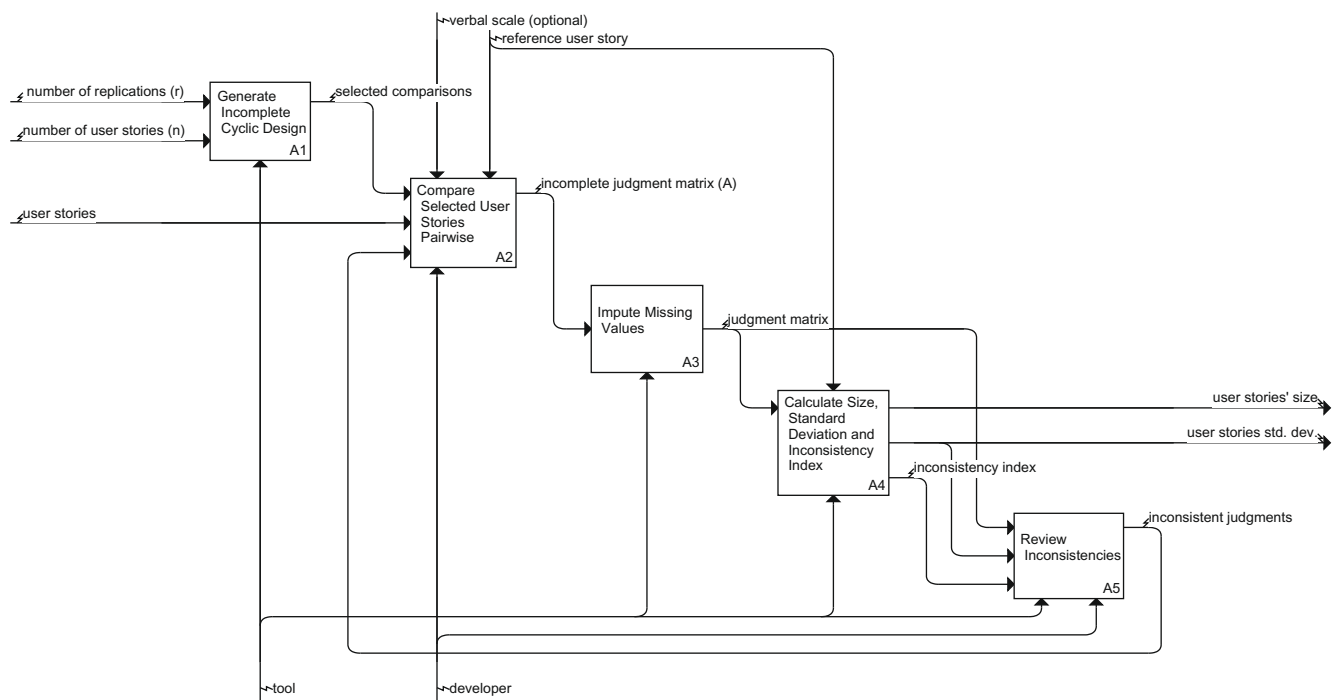
$$sp_i = \begin{cases} 23.02 \\ 5.85 \\ 13.91 \\ 5.0 \end{cases}$$

The total size for the project is 47.78 story points, with an Inconsistency Index of 0.94 which yields a size range of  $\pm 47\%$  for each user story (8).

As the estimator is not happy with such a range, he decides to review his judgments. To do this, he resorts to Eq. (1), which states that, in a perfectly consistent matrix  $a_{ij} \times a_{jk} = a_{ik}$ , that is, the relative size of user story<sub>i</sub> with respect to user story<sub>j</sub> multiplied by the relative size of user story<sub>j</sub> with respect to user story<sub>k</sub> must be equal to the relative size of user story<sub>i</sub> with respect to user story<sub>k</sub>. To

**Table 1**  
Results of the Burton experiments.

Number of comparisons in which each concept was included (r)	Number of comparisons with respect to the complete design (%)	Number of comparisons in the Full Factorial design	Number of comparisons in the ICD	Lowest correlation with results from the complete design	Mean correlation with results from the complete design	Comments
2	10	210	21	0.46	0.79	Mean correlation is acceptable, but worst-case correlation is too low
4	20		42	0.58	0.95	Mean correlation and worst-case correlation are acceptable
6	30		63	0.80	0.96	Almost as good as the complete design
8	40		84	0.97	0.98	



**Fig. 4.** The Fractional Paired Comparison method. Each user story is only compared to those indicated by the ICD.

operationalize this concept, we divide Eq. (1) by  $a_{ik}$  and obtain (10). The amount by which Eq. (10) differs from 1 when the actual judgments are plugged in is used to identify the largest inconsistent judgments (see Fig. 3).

$$\frac{a_{ij} \times a_{jk}}{a_{ik}} = 1 \quad (10)$$

After reviewing the estimates, the estimator decides that A is half the size of C and not five times larger as previously stated. So, he records the new judgment in the judgment matrix (11).

$$A^{4 \times 4} = \begin{matrix} & \begin{matrix} A & B & C & D \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \end{matrix} & \begin{bmatrix} 1 & 3.0 & .50 & 2.0 \\ .33 & 1 & .25 & 1.5 \\ 2.0 & 4.0 & 1 & 5.0 \\ .50 & .67 & .20 & 1 \end{bmatrix} \end{matrix} \quad (11)$$

This change results in the reduction of the Inconsistency Index from 0.94 to 0.27. The lower value indicates a more consistent evaluation of the relative size of the user stories. The new Inconsistency Index yields a size range for each user story of  $\pm 14\%$ , and the estimator decides to accept the results.

The new total size for the project is 48.7 story points, with individual sizes and standard deviations of:

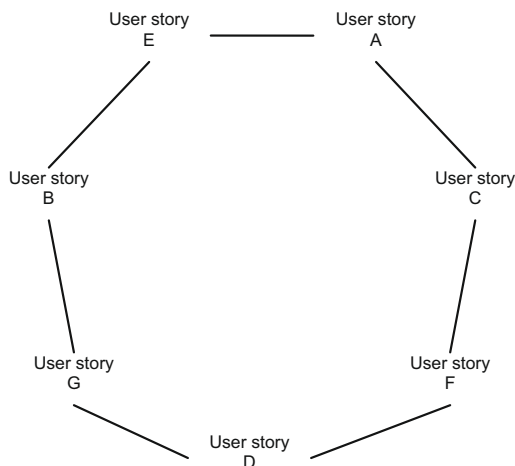
$$sp_i = \begin{cases} 13.1 \pm 1.83 \\ 5.9 \pm 0.83 \\ 24.7 \pm 3.46 \\ 5.0 \pm 0.70 \end{cases}$$

#### 4. Reducing the number of comparisons with Incomplete Cyclic Designs (ICD)

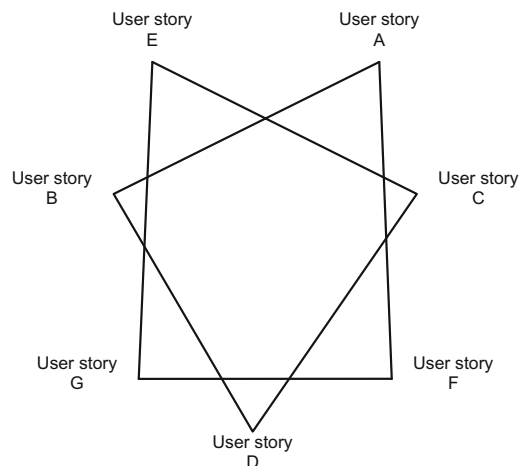
##### 4.1. Overview

The Full Factorial Paired Comparison method provides a solution to the problem of dealing with inconsistent judgments. The problem is that the method does not scale up as the number of comparisons required grows with the square of the number of user stories being estimated.

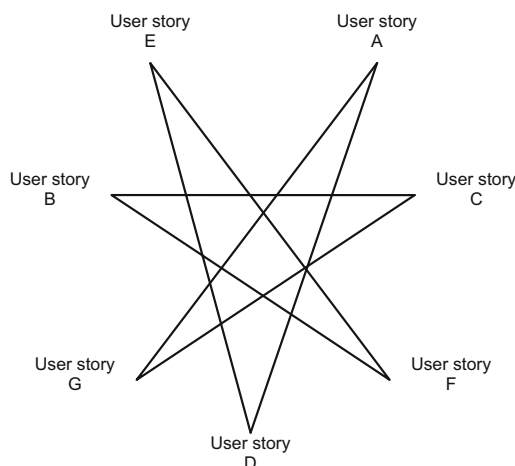
To solve it, several authors [9,10,15] had proposed the use of Incomplete Cyclic Designs (ICDs) to select a subset of the  $n(n-1)/2$  comparisons required by the Full Factorial method.



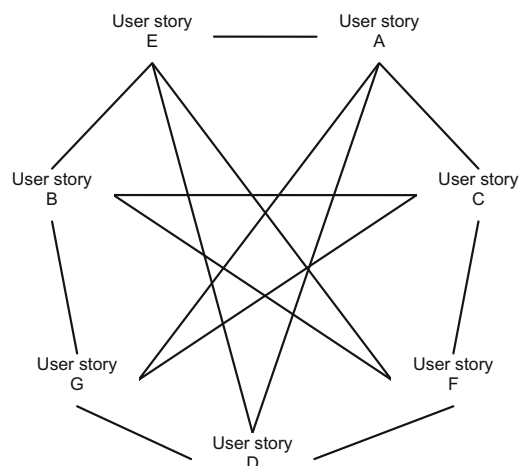
(a)  $r = 2, s = 1$



(b)  $r = 2, s = 2$



(c)  $r = 2, s = 3$



(d)  $r = 4, s = 1$

**Fig. 5.** Four different Incomplete Cyclic Designs. Note 1: Designs a, b, and c are created by increasing the distance ( $s$ ) between user stories. Note 2: Design d is the result of merging designs a and c.

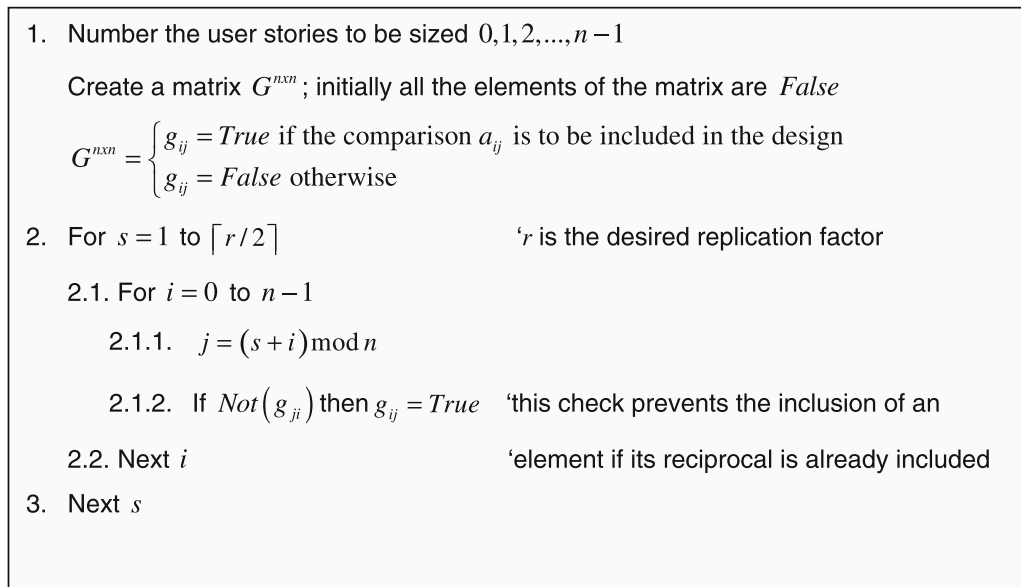


Fig. 6. ICD-generating algorithm.

Such designs are called fractional designs because they contain only a fraction of all possible comparisons. The ICD technique is used to select which stories are to be compared. Starting with one user story, successive comparisons are selected in a cyclical fashion using the arithmetic modulo  $n$ . This method is further developed in the following sections.

Table 1 shows the results of a series of experiments conducted by Burton [8] to evaluate the impact of a reduced number of comparisons in the reliability of the estimates. The experiment consisted of evaluating the correlation between the results of a Full Factorial estimation with the results of a number of fractional (ICD) designs, each with a different  $r$  for two groups with multiple respondents.

The closer to 1 the correlation between the Full Factorial and the fractional designs and the lower the spread between the lowest and the mean correlations, the more accurate and precise the values estimated using the ICD were.

The dataset used in the experiment consisted of 21 different concepts for which the test subjects needed to quantify their semantic similarity. The Full Factorial design required 210 comparisons. The experiment showed that a fractional design with only 63 comparisons displayed a high correlation (0.80–0.96) between the similarities estimated by the two methods.

#### 4.2. The Fractional Paired Comparison method

The Fractional Paired Comparison method requires that: (1) we decide which comparisons to make and (2) we compensate for the

missing values. This adds two new activities to the original process: Generate Incomplete Cyclic Designs and Impute Missing Values (see Fig. 4), which are explained in later sections.

#### 4.3. Generating Incomplete Cyclic Designs (ICD)

The proposed Incomplete Design Cycle (ICD) construction process starts by arranging, in a random order, the user stories along a circle, and joining adjacent user stories with a line. Each line corresponds to a comparison.

The design generated in this way (see Fig. 5a) consists of a total of 7 comparisons, with each user story appearing in two comparisons, one with the user story to its left and the other with the one to its right. The design's distance  $s$  is the minimum number of hops along the circle needed to reach the stories being compared. In Fig. 5a,  $r = 2$  and  $s = 1$ .

Additional designs (see Fig. 5b and c) are generated by increasing the distance between the user stories compared. ICDs with a higher  $r$  are obtained by merging simpler designs, as shown in Fig. 5d, which results from the juxtaposition of the designs in Fig. 5a and c.

To operationalize the generation of ICD, we use an adjacency matrix  $G^{n \times n}$  and the algorithm in Fig. 6. Table 2 shows the matrix representation of the designs in Fig. 5.

#### 4.4. Imputing missing values

Before we can calculate the size of the user stories, we need to impute the missing values of the judgment matrix, that is, those corresponding to the comparisons that were skipped by the design, with a representative value. Note that the assignments  $a_{ii} = 1$  and  $a_{ji} = 1/a_{ij}$ , the elements on the principal diagonal and the reciprocal of the judgments, respectively, are not missing values and should be made before the imputing calculations are performed.

As the comparisons to be skipped were selected at random by the ICD construction procedure, we can impute them with the mean value [16,17] of the row to which they would have belonged using the algorithm in Fig. 7.

Because the number of judgments made in an ICD with replication factor  $r$  is lower than in the case of the Full Factorial design, we need to change the formula for the Inconsistency Index to reflect

**Table 2**  
Matrix representation of the designs in Fig. 5.

User story	A	C	F	D	G	B	E
A		a, d	b	c, d			
C			a, d	b	c, d		
F				a, d	b	c, d	
D					a, d	b	c, d
G	c, d					a, d	b
B	b	c, d					a, d
E	a, d	b	c, d				

(a)  $r = 2$ ,  $s = 1$ ; (b)  $r = 2$ ,  $s = 2$ ; (c)  $r = 2$ ,  $s = 3$ ; and (d)  $r = 4$ .

Assume the existence of a judgment matrix  $A^{n \times n}$  filled according to the ICD represented by the matrix  $G^{n \times n}$  whose missing  $a_{ij}$  values are zero

```

1. For  $i = 1$  to  $n$ 
  1.1.  $meanvalue = 1$ ;  $c = 0$ ;
  1.2. For  $j = 1$  to  $n$ 
    1.2.1. If  $a_{ij} \neq 0$  then
      1.2.1.1.  $meanvalue = meanvalue \times a_{ij}$ 
      1.2.1.2.  $c = c + 1$ 
    1.2.2. Endif
  1.3. Next  $j$ 
  1.4.  $meanvalue = meanvalue^{\frac{1}{c-1}}$  'this is the row's geometric mean, the  $c-1$  in
                                     'the denominator is used to compensate for the
                                     'fact that an element is always equal to itself
  1.5. For  $j = 1$  to  $n$ 
    1.5.1. If  $a_{ij} = 0$  then 'we assume zero to be a missing value
      1.5.1.1.  $a_{ij} = meanvalue$ 
      1.5.1.2.  $a_{ji} = 1 / meanvalue$ 
    1.5.2. Endif
  1.6. Next  $j$ 
2. Next  $i$ 

```

Fig. 7. Imputation algorithm.

the reduced number of degrees of freedom. To do this, we substitute in the denominator in (5) the number of judgments required by the Full Factorial –  $(n(n-1)/2)$  – with the number of judgments required by an ICD with a replication factor of  $r - (r \times n/2)$ .

$$InconsistencyIndex = \sqrt{\frac{2 \sum_{i < j}^n \left( \ln a_{ij} - \ln \frac{mrs_i}{mrs_j} \right)^2}{rn - 2n + 2}} \quad (12)$$

The computation of the standard deviation of the size of the user stories (8) remains unchanged.

## 5. Empirical verification

In this section, we compare the performance of the Fractional Paired Comparison method with the results generated by the Full Factorial method using a set of 15 user stories (see Table 3) derived

Table 3

User stories derived from the job board [18].

Dataset (1 or 2)	Role	Action	Benefit
1	Job seeker	Login	I can use the system capabilities reserved for registered job seekers
1		Logout	...to end a session and protect my data from being accessed by unauthorized people
1, 2		Register	I can make my data available to headhunters and use the system capabilities reserved for registered job seekers
1		Search job announcements	I can find selected postings based on keywords or criteria, such as job category, location, industry, and city
1	Recruiter	Create career alert	I will get email notifications whenever a new announcement matching the search criteria is first posted
2		Suspend career alert	I will not receive notifications without deleting the career alert
2		Delete career alert	I will not receive further notifications
1		Upload resume	It can be searched and read by recruiters
2		Delete resume	It is not longer available to recruiters
2		Login	I can use the system capabilities reserved for registered recruiters
2		Logout	...to end a session and protect my data from being accessed by unauthorized people
2		Post	I can post a new job announcement
2	System owner	Edit	I can modify an existing job announcement
2		Delete	I can delete an existing job announcement
1, 2		Notify	I can email all job seekers re new postings, according to their career alert status

from a popular Canadian job board [18]. The user stories are described in Table 3 using the template: “As <role> I would like to <action> so that <benefit>”.

For an evaluation of the paired comparison method against actuals, refer to [7,19,20].

### 5.1. Method set-up

The verification was conducted using two subsets of different sizes to explore the impact of the number of user stories in the reliability of the fractional method. As we only obtained 15 user stories from the website, two of them were included in both sets. The same user story was used as a reference to avoid differences originating from the use of different references. In Table 3, the numbers in the left-hand column indicate which user stories were included in which dataset (1 or 2, or both).

First, a Full Factorial estimation was performed on each dataset by a senior software developer. From these two datasets, the fractional designs were generated by excluding from the calculations those values that would have been skipped by the ICDs. This approach allowed us to control for the judgment errors associated with repeated questioning.

Fig. 8 shows the data and the comparisons included in the first dataset for three different designs with  $r = 6$  (full factorial – 100% of the comparisons),  $r = 4$  (66% of the comparison), and  $r = 2$  (33% of the comparisons). In Fig. 8, only the shaded values need to be provided by the estimator. The user story ‘Notification’ is the reference user story, and its predetermined size is 10 story points.

The Full Factorial design required 21 comparisons, while the first and second ICDs required 14 and 7, respectively. The results are shown in Table 4.

Story Points	User Story	Registration	Notification	Create Alert	Search jobs	Login (job seeker)	Upload resume	Logout (job seeker)
	Registration		1.5	2	3			
10	Notification			1.5	2	3		
	Create alert				1.5	2	5	
	Search jobs					1.5	3	4.0
	Login job seeker	0.25					2.0	2.5
	Upload resume	0.11	0.17					1.2
	Logout job seeker	0.10	0.14	0.17				

(a) Full Factorial design —  $r = 6$

Story Points	User Story	Registration	Notification	Create Alert	Search jobs	Login (job seeker)	Upload resume	Logout (job seeker)
	Registration		1.5	2				
10	Notification			1.5	2			
	Create alert				1.5	2		
	Search jobs					1.5	3	
	Login job seeker						2.0	2.5
	Upload resume	0.11						1.2
	Logout job seeker	0.10	0.14					

(b) Fractional design —  $r = 4$

Story Points	User Story	Registration	Notification	Create Alert	Search jobs	Login (job seeker)	Upload resume	Logout (job seeker)
	Registration		1.5					
10	Notification			1.5				
	Create alert				1.5			
	Search jobs					1.5		
	Login job seeker						2.0	
	Upload resume							1.2
	Logout job seeker	0.10						

(c) Fractional design —  $r = 2$

**Fig. 8.** Empirical verification using three different designs for dataset 1. Note 1: Fig. 8a design with  $r = 6$  (full factorial – 100% of the comparisons). Note 2: Fig. 8b design with  $r = 4$  (66% of the comparisons). Note 3: Fig. 8c design with  $r = 2$  (33% of the comparisons).



**Table 4**

Dataset 1: estimation results of 7 user stories for 3 different replication factors.

User story	<i>r</i> = 6 (Full Factorial) 21 Comparisons		<i>r</i> = 4 14 Comparisons		<i>r</i> = 2 7 Comparisons	
	Estimated story points	Std. dev.	Estimated story points	Std. dev.	Estimated story points	Std. Dev.
Registration	14.4	0.3	18.8	4.97	31.9	20.86
Notification (reference)	10.0	0.2	10.0	2.65	10.0	6.54
Create Alert	7.4	0.17	5.7	1.51	6.8	4.45
Search jobs	5.0	0.11	4.8	1.27	7.5	4.9
Login job seeker	3.4	0.08	4.5	1.19	9.0	5.88
Upload resume	1.6	0.04	3.0	0.79	8.6	5.62
Log out job seeker	1.3	0.0	2.7	0.71	8.0	5.23
Project total	43.1		49.4		81.7	
Inconsistency Index	0.06		0.70		1.73	

The second dataset consisted of 10 user stories, which were estimated using a Full Factorial design and 4 different ICDs. The results are shown in Table 5.

Note that, for both datasets, the minimal replication ICDs, *r* = 2, correspond to the simple triangulation procedure proposed in the Agile literature.

**Table 5**

Dataset 2: estimation results of 10 user stories for 5 different replication factors.

User story	<i>r</i> = 9 (Full Factorial) 90 Comparisons		<i>r</i> = 8 40 Comparisons		<i>r</i> = 6 30 Comparisons		<i>r</i> = 4 20 Comparisons		<i>r</i> = 2 10 Comparisons	
	Estimated story points	Std. dev.	Estimated story points	Std. dev.	Estimated story points	Std. dev.	Estimated story points	Std. dev.	Estimated story points	Std. dev.
Registration	13.0	0.74	13.5	1.49	14.2	2.69	16.4	4.20	29.7	17.94
Notification (reference)	10.0	0.57	10.0	1.11	10.0	1.90	10.0	2.56	10.0	6.04
Post announcement	6.1	0.35	6.0	0.66	5.2	0.99	4.1	1.05	5.4	3.26
Edit announcement	4.4	0.25	4.3	0.48	3.7	0.70	3.6	0.92	6.7	4.05
Login recruiter	3.0	0.17	2.8	0.31	2.7	0.51	3.1	0.79	7.0	4.23
Suspend alert	2.3	0.13	2.5	0.28	2.5	0.47	3.3	0.85	7.8	4.71
Delete alert	1.9	0.11	2.1	0.23	2.7	0.51	3.8	0.97	8.5	5.13
Delete announcement	1.5	0.09	1.7	0.19	2.4	0.46	3.8	0.97	8.4	5.07
Delete resume	1.3	0.07	1.5	0.17	2.2	0.42	3.7	0.95	8.7	5.25
Logout recruiter	1.0	0.06	1.2	0.13	1.8	0.34	3.1	0.79	7.5	4.53
Project total	44.6		45.7		47.5		54.9		99.7	
Inconsistency Index	0.18		0.35		0.60		0.81		1.91	

**Table 6**

Method evaluation at the user story level.

Replication factor ( <i>r</i> )	Inconsistency Index	<i>MMRE</i> <sup>a</sup>	<i>Pred</i> (0.25) <sup>a</sup>	Comments
<i>First dataset – 7 user stories</i>				
4	0.70	0.45	0.33	MMRE and PRED do not meet the established criteria
2	1.73	2.10	0.17	Reducing the Inconsistency Index would help improve both measures, but, under the stated conditions, the method does not produce acceptable results at the user story level
				Notice that the experiment with <i>r</i> = 2 corresponds to the triangulation approach recommended in the Agile literature
<i>Second dataset – 10 user stories</i>				
8	0.35	0.09	1	Estimates for individual user stories are acceptable
6	0.60	0.34	0.56	As the number of comparisons is reduced, the MMRE and PRED for individual user story estimates start to deteriorate
4	0.81	0.84	0.33	
2	1.91	2.85	0.11	Estimates for individual user stories are not acceptable. Note that this is the case for triangulation against 2 other user stories

<sup>a</sup>The denominator used in both calculations is the number of user stories – 1, to account for the reference element.

$$MMRE = \frac{\sum abs(x_i - \hat{x}_i)/x_i}{n - 1}$$

Pred(0.25) = *k*/(*n* – 1) is the proportion of observations (*k*) that fall within 25% of the actual.

## 5.2. Discussion of results

Each ICD's performance was evaluated using the Mean Magnitude Relative Error (*MMRE*) and the Predictive Quality Indicator (*Pred*) at the user story level and at the project levels. At the user story level, *MMRE* and *Pred* were calculated by comparing the estimated size of the user stories for a given replication against the value estimated by the Full Factorial method, and, at the project level, we compared the sum of the sizes of all the user stories in the project.

Evaluating the method on these two levels was important, because the planning of each iteration requires not only that the overall project size be reliable, but also that the estimation of each user story be acceptable.

The results were considered acceptable when their *MMRE* was less than or equal to 0.25 and their *Pred*(0.25) was greater than or equal to 0.75 [6,21].

Tables 6 and 7 summarize the performance for the estimation results presented in Tables 4 and 5. The values obtained show that the fractional designs produce acceptable results at the project level with very low *r* (*r* = 4), but that good estimates at the user story level required higher degrees of replication or lower Inconsistency Index values. As expected, the influence of inconsistent judgments increased with a reduction in the number of comparisons, and this resulted in higher *MMRE*s and lower *Pred*(0.25)s in the experimental situation.



**Table 7**

Method evaluation at the project level.

Projects included in the calculation	MMRE	Pred(0.25)	Comments
All projects	0.43	0.67	MMRE and PRED do not meet the stated criteria
Excluding projects with $r = 2$	0.11	1	By excluding the projects with a low replication factor, MMRE and PRED are brought to acceptable levels. Estimates produced with $r > 2$ are adequate at the project level
Only projects with $r = 2$	1.06	0	This corroborates the comments made above

Note that, as a consequence of retaining the values from the Full Factorial design to control for judgment error in the experiment design, we did not correct any values to obtain an Inconsistency Index closer to the recommended 0.35. Had we allowed ourselves to perform one or two amendments, as we would normally do in practice, we would have brought the index down and improved the MMREs and Pred(0.25)s in the low-replication estimations. Reasoning along this line, triangulating against two other user stories ( $r = 2$ ) would require almost perfect consistency on the judgments rendered for the extra comparisons to increase the estimate's reliability.

## 6. Summary

Agile estimation approaches usually start by sizing the user stories to be developed by comparing them to one another. Various techniques, with varying degrees of formality, have been proposed by the Agile community to conduct the comparisons – plain contrasts, triangulation, planning poker, and voting. This article adds to these techniques by proposing the use of a modified paired comparison method, in which a reduced number of comparisons is selected according to an Incomplete Cyclic Design.

An empirical verification of this proposal was conducted using two datasets, showing that the proposed method produces good estimates, even when the number of comparisons is reduced by half those required by the original formulation of the paired comparison method. A byproduct of the evaluation is the conclusion that the simple triangulation advocated in the Agile literature does not to automatically result in more reliable estimates. Low-replication comparisons require a high degree of consistency among judgments.

To confirm these results the authors plan to conduct follow-up studies in their respective organizations.

Although we have used story points to illustrate the article, the techniques described could be equally applied using different size units such as lines of code or to the estimation of durations using ideal days or effort.

Those seeking to introduce the method in their organizations must be aware that, while people readily buy into the idea of comparing user stories to one another, they tend to become discouraged by the underlying mathematics. Therefore, two things are required for a successful deployment of the method: first, the careful selection of the number of details to be included in training presentations and process documentation; and second, the development of a simple spreadsheet to support these calculations.

## Acknowledgment

We would like to thank Ahmed Bedhief for patiently going through all the comparisons required by the Full Factorial experiment.

## References

- [1] <[http://en.wikipedia.org/wiki/Story\\_points](http://en.wikipedia.org/wiki/Story_points)>, "Story Points," Wikipedia, 4/5/2009.
- [2] N. Dalkey, The Delphi Method: An Experimental Study of Group Opinion, Rand Corporation, Santa Monica, 1969.
- [3] M. Cohn, Agile Estimating and Planning, Prentice Hall, Upper Saddle River, NJ, 2006.
- [4] M. Cohn, Tutorial on agile estimating and planning, Agile 2006 Conference, Mountain Goat Software, Minneapolis, 2006.
- [5] G. Bozoki, An expert judgment based software sizing model, Journal of Parametrics XIII (May) (1993).
- [6] E. Miranda, Improving subjective estimates using paired comparisons, IEEE Software 18 (January/February) (2001) 87–91.
- [7] M. Shepperd, M. Cartwright, Predicting with sparse data, IEEE Transactions on Software Engineering 27 (November) (2001) 987–998.
- [8] M.L. Burton, Too many questions? The uses of incomplete cyclic designs for paired comparisons, Field Methods 15 (May) (2003) 115–130.
- [9] I. Spence, Incomplete experimental designs for multidimensional scaling, in: R. Golledge, J. Rayner (Eds.), Proximity and Preference: Problems in the Multidimensional Analysis of Large Data Sets, University of Minnesota Press, Minneapolis, 1982, pp. 29–45.
- [10] I. Spence, D. Domoney, Single subject incomplete designs for nonmetric multidimensional scaling, Psychometrika 39 (1974).
- [11] J. Barzilai, W. Cook, B. Golany, Consistent weights for judgments matrices of the relative importance of alternatives, Operations Research Letters 6 (July) (1987).
- [12] G. Crawford, C. Williams, A note on the analysis of subjective judgment matrices, Journal of Mathematical Psychology 29 (1985) 387–405.
- [13] J. Aguaron, J. Moreno-Jimenez, The geometric consistency index: approximated thresholds, European Journal of Operational Research 147 (2003) 137–145.
- [14] J. Hihn, K. Lum, Improving Software Size Estimates by Using Probabilistic Pairwise Comparison Matrices, in: 10th IEEE International Symposium on Software Metrics (METRICS'04), 2004, pp. 140–150.
- [15] H. David, Cyclic designs, in: S. Kotz (Ed.), Encyclopedia of Statistical Sciences, 2007, John Wiley & Sons, 1965.
- [16] I. Myrtveit, E. Stensrud, U. Olsson, Analyzing data sets with missing data: an empirical evaluation of imputation methods and likelihood-based methods, IEEE Transactions on Software Engineering 27 (November) (2001) 999–1013.
- [17] Q. Song, M. Shepperd, M. Cartwright, A short note on safest default missingness mechanism assumptions, Empirical Software Engineering 10 (April) (2005) 235–243.
- [18] <<http://www.workopolis.com>>, "Job Board," Toronto Star Newspapers, 2008.
- [19] G. Bozoki, Performance simulation of SSM, in: ISPA 13th Annual Conference, 1991.
- [20] E. Miranda, Evaluation of the paired comparisons method for software sizing, in: 22nd International Conference on Software Engineering, Limerick, Ireland, 2000, pp. 597–604.
- [21] S. Conte, H. Dunsmore, V. Shen, Software Engineering Metrics and Models, Benjamin-Cummings Publishing Company, 1986.

## **ANNEX II**

### **Protecting Software Development Projects against Underestimation**

# Protecting Software Development Projects Against Underestimation

**Eduardo Miranda**, *École de Technologie Supérieure, Université du Québec, Montreal, Canada*  
**Alain Abran**, *École de Technologie Supérieure, Université du Québec, Montreal, Canada*

## ABSTRACT ■

When a project in progress has been seriously underestimated, it is essential to figure out how much additional effort is required to complete it within its original scope and delivery date. This article posits that project contingencies should be based on the amount it will take to recover from the underestimation, and not on the amount that would have been required had the project been adequately planned from the beginning, and that these funds should be administered at the portfolio level. A model to calculate the required funds is developed.

**KEYWORDS:** risk management; contingency funds; management reserves; project allowances; cost of recovery; project estimation; MAIMS; probabilistic cost analysis

## INTRODUCTION ■

According to the Project Management Institute (PMI), a “contingency reserve” is “the amount of funds, budget, or time needed above the estimate to reduce the risk of overruns of project objectives to a level acceptable to the organization” (PMI, 2004, p. 355). Contingency funds are meant to cover a variety of possible events and problems that are not specifically identified or to account for a lack of project definition during the preparation of planning estimates. When the authority for the use of the funds is above the project management level, it receives the name of management reserve.

In practice, contingencies are added to projects using heuristics such as the 10% or 20% of the project budget or by accruing percentage points on the basis of responses given to a risk questionnaire. More mature organizations might even run Monte Carlo simulations to calculate expected values. Whatever the approach chosen, in deciding how much and how to administer the contingency funds, one cannot ignore the human and organizational considerations that dictate decision making in real-world projects. Specifically, one needs to consider management preference of schedule over cost, time preferences, and the money-allocated-is-money-spent behavior (Kujawski, Alvaro, & Edwards, 2004).

A good example of the preference for schedule over cost is given by Stephen Grey (1995): “While most people will be willing to accept that cost could exceed expectations, and might even take a perverse delight in recounting past examples, the same is not true for deadlines. This is probably due to the fact that cost overruns are resolved in-house, while schedule issues are open and visible to the customer” (p. 108). In other words, project delays and scope cuts are not great career builders, so when faced with a schedule overrun, management’s preferred course of action is not to replan to achieve the best economic outcome but to attempt to keep the schedule by adding people, despite the fact that adding resources midway through a project will result in one or more of the following (Sim & Holt, 1998):

- need to break down the work into additional segments, so that they can be allocated to the newcomers;
- need to coach the new staff;
- additional integration work; and
- additional coordination effort.

This means that if we know that contingency funds will be used first and foremost to maintain a schedule and not just to pay for underestimated work, we should acknowledge in their calculation the extra cost incurred by the above activities.

Wishful thinking (Babad & Katz, 1991) and inaction inertia (Tykocinski, Pittman, & Tuttle, 1995) are examples of time preferences that result in postponing the acknowledgment of a delay until the last possible moment. Todd

Project Management Journal, Vol. 39, No. 3, 75–85  
 © 2008 by the Project Management Institute  
 Published online in Wiley InterScience  
 (www.interscience.wiley.com)  
 DOI: 10.1002/pmj.20067

# Protecting Software Development Projects Against Underestimation

Little (2006) commented on the unwillingness to acknowledge project delays: "This is the result of the project manager holding on to a deadline in hopes that a miracle will occur and the software will release. Finally the day of reckoning occurs, with no miracle in sight. At this point, the project estimate is usually reset. In many cases, this cycle repeats until the software releases" (p. 52).

The tendency to procrastinate should also be factored into the calculation of contingency funds because, other things being equal, the later the underestimation is acknowledged, the higher the number of people required and, consequently, the higher the cost.

These two premises led us to postulate that:

*ContingencyFunds* =

$$\iint \text{RecoveryCost}(u, t) P(t) p(u) dt du \quad (1)$$

Equation 1 ascertains that contingency funds must equal the expected recovery cost of a project—that is, the

effort necessary to recover from an underestimation of magnitude  $u$  upon which we act at time  $t$  by the probability of  $u$  and the probability of  $t$ .

Having considered management predilection for schedule over budget and the time preferences, it is time now to look at the third behavior that affects the use of contingency funds: the money-allocated-is-money-spent (MAIMS) (Gordon, 1997; Kujawski et al., 2004) behavior. The MAIMS behavior implies that, for a variety of reasons, once a budget is allocated it will tend to be spent in its entirety, and, as a consequence, cost underruns are seldom available to offset overruns. This negates the basic premise that contingency usage is probabilistic and so, managing the funds above the project level becomes the obvious and mathematically valid solution for its effective and efficient administration.

The remainder of the article defines and provides a rationale for *RecoveryCost*( $u, t$ ),  $P(t)$  and  $P(u)$  and

explains how these functions can be calculated in practice. We also present a numerical solution to Equation 1 and explain why contingency funds should be administered above the project level.

## The Recovery Effort

Figure 1 illustrates the effort makeup of a project under recovery assuming that the objective is to preserve the original scope and the delivery date to which a commitment has been made:

- The budgeted effort ( $E_b$ ) is the amount of effort originally allocated to the project and is the product of the time budgeted ( $T_b$ ) and the original staff ( $FTE_b$ ).
- $t$  is the time at which the underestimation is acknowledged and a decision to do something about it is finally made.
- $T_a$  is the mean time between when the decision to bring in new people was made and the time at which the new staff arrives.

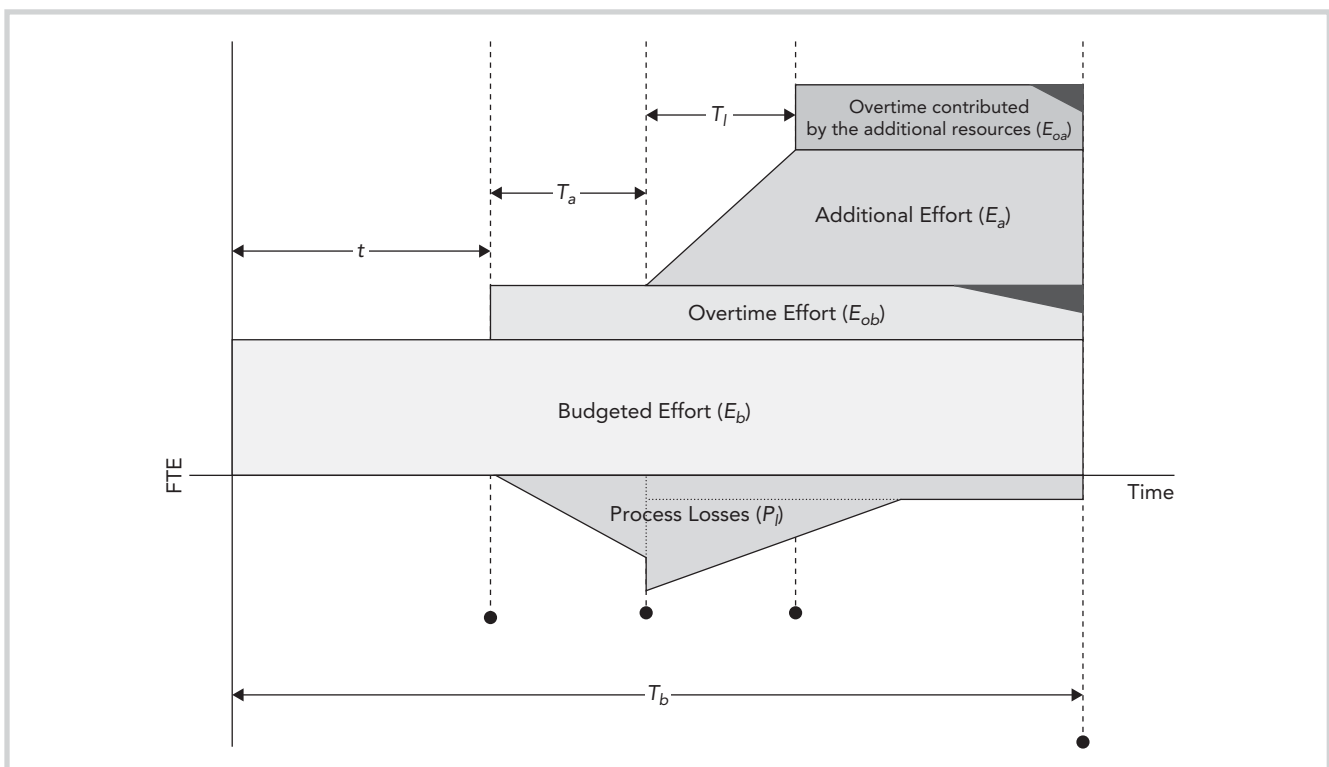


Figure 1: Recovery cost of an underestimated project [adapted from Grey, 1995].

- The additional effort ( $E_b$ ) is the effort that will be contributed by the resources brought in to help recover from the delay. The sloped left side of the quadrilateral models the fact that there will be a certain time interval ( $T_l$ ) before the new staff becomes fully productive.
- The overtime efforts ( $E_{ob}$  and  $E_{oa}$ ) are the efforts contributed through overtime by both the original and the additional resources. Overtime efforts are affected by fatigue, as modeled by the dark triangles on the upper-right corners of the corresponding rectangles.
- The process losses ( $P_l$ ) include all the extra effort: ramp-up, coaching, and communication overhead imposed on the original staff by newcomers.

The simplicity of this makeup is deliberate. While other effort breakdowns are certainly possible, these would come at the expense of more complicated expressions, perhaps based on hypothesized parameters, which would make the model harder to explain. A complete list of the model's parameters is provided in the Appendix

#### Calculating the Number of Additional People— $FTE_a(t, u)$

Mathematically, the effort required to recover from an underestimation ( $u$ ) would be equal to the effort that could be contributed through the overtime of the original staff ( $E_{ob}$ ), plus the effort of those brought in to help ( $E_a$ ) and their overtime ( $E_{oa}$ ), less the effort necessary to compensate for the process losses ( $P_l$ ).

$$u = E_a + E_{ob} + E_{oa} - P_l \quad (2)$$

The effort contributed by the additional staff would then be:

$$E_a = FTE_a \times (T_b - t - T_a) - \frac{FTE_a T_l}{2} \quad (3)$$

The term  $\frac{FTE_a T_l}{2}$  accounts for the learning effort of the new staff.

The effort available through overtime is modeled as the percentage of the overall available effort less a

productivity loss due to fatigue arising at a *lag* time after the decision to utilize overtime has been made:

$$E_{ob} = C_o FTE_b \left[ T_b - t - \frac{C_d (T_b - t - lag)^2}{2} \right] \quad (4)$$

$$E_{oa} = C_o FTE_a \left[ T_b - t - T_a - T_l - \frac{C_d (T_b - t - T_a - T_l - lag)^2}{2} \right] \quad (5)$$

The constants  $C_o$  and  $C_d$  stand for the maximum amount of overtime to be used in the project and the rate of productivity decay after *lag* weeks of working overtime, respectively.

The process losses will be modeled as:

$$P_l = \frac{C_r \times FTE_a \times T_l}{2} + \frac{C_c \times FTE_a \times T_l}{2} + \frac{C_i \times FTE_a (FTE_a + 2 \times FTE_b - Teams) (T_b - t - T_a)}{2 \times Teams} \quad (6)$$

The constants  $C_r$ ,  $C_c$ , and  $C_i$  stand for the ramp-up, coaching, and interaction factors, respectively. *Teams* is the number of groups into which the work is organized. Justification for these choices, together with that for  $C_d$ , will be given later.

Substituting the terms  $E_a$ ,  $E_{ob}$ ,  $E_{oa}$ , and  $P_l$  in Equation 2 by Equations 3–6 we obtain:

$$\begin{aligned} & FTE_a^2 \left[ \frac{C_i (t + T_a - T_b)}{2 Teams} \right] \\ & + \frac{FTE_a}{2 Teams} \left[ -Teams \left[ \frac{2t + 2T_a + T_l + C_c T_l + C_r T_l + 2C_o (t + T_a + T_l - T_b)}{2} \right. \right. \\ & \quad \left. \left. + C_d C_o (lag + t + T_a + T_l - T_b)^2 - 2T_b \right] + C_i (2 FTE_b - Teams) (t + T_a - T_b) \right] \\ & - \frac{FTE_b [2C_o t + C_d C_o (lag + t - T_b)^2 - 2C_o T_b]}{2} - u = 0 \end{aligned} \quad (7)$$

Equation 7 is an equation of the second degree, the general solution of which is:

$$FTE_a = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

So, by renaming

$$a = \frac{C_i (t + T_a - T_b)}{2 Teams}$$

$b(t) =$

$$\left[ \frac{-Teams \left[ \frac{2t + 2T_a + T_l + C_c T_l + C_r T_l + 2C_o (t + T_a + T_l - T_b)}{2} \right. \right. + C_i (2 FTE_b - Teams) (t + T_a - T_b) \left. \left. \right]}{2 Teams} \right]$$

$$c(u, t) = \frac{FTE_b [2C_o t + C_d C_o (lag + t - T_b)^2 - 2C_o T_b]}{2} - u$$

we obtain:

$$E_a = FTE_a \times (T_b - t - T_a) - \frac{FTE_a T_l}{2} \quad (8)$$

# Protecting Software Development Projects Against Underestimation

This is the number of additional resources required to recover from an underestimation of magnitude  $u$  acknowledged at time  $t$ .

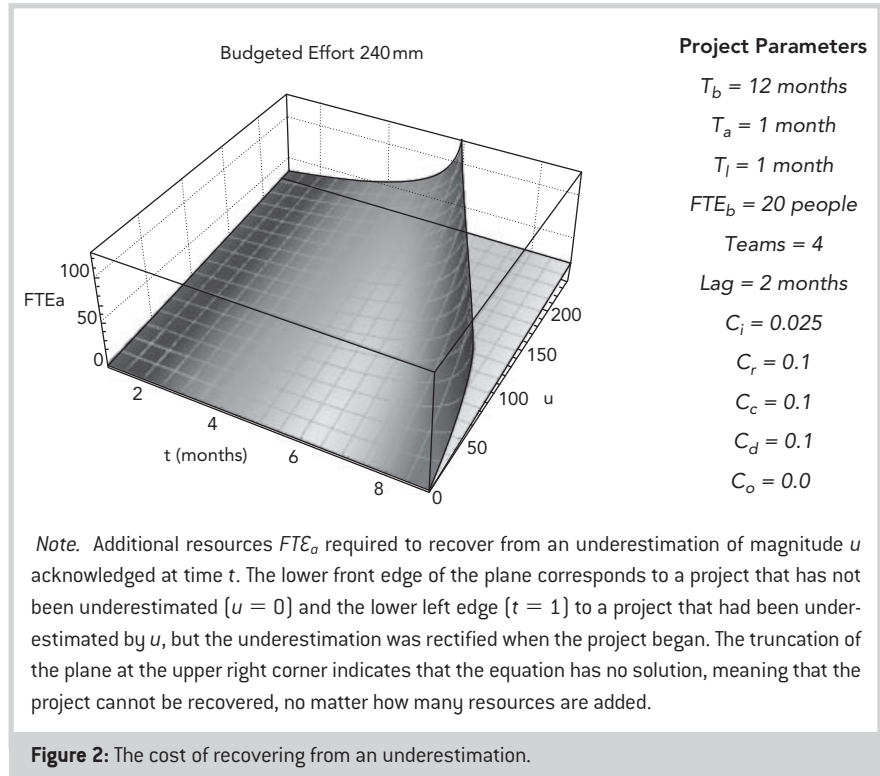
## Project Example

The solution space for Equation 8 is illustrated in Figure 2. The vertical axis,  $FTE_a(t, u)$ , is the number of resources to be added to the project, the lower axis,  $t$ , corresponds to the time at which the underestimation is acknowledged, and the third axis,  $u$ , is the magnitude of the underestimation in man-months. The upper plane in the figure reflects the number of additional resources needed to recover from the underestimation while keeping the original project's completion date fixed or constant. If we did not apply this constraint, the upper plane surface could have a totally different shape. The lower plane shows the number of additional resources required had the underestimated work been included in the original plan. All project parameters are listed on the right-hand side of the figure.

The function is valid only if additional resources are required, that is (a), if the underestimation is greater than the extra effort that can be provided through the use of overtime alone and if the decision to bring in the additional resources is made on time and (b) if the process losses are greater than the effort that could be generated in the remaining time ( $T_b - t - T_a$ ). Then the equation has no real solution: it is an imaginary number, indicating that under the circumstances it would be impossible to maintain the schedule, no matter how many people are added.

## Process Losses

Frederick Brooks (1995) coined the well-known admonition that adding an extra person to a late project made it later, which even if a little extreme, has some element of truth. Adding people midway through a project creates additional work (the process losses) that would have not existed



**Figure 2:** The cost of recovering from an underestimation.

otherwise. The process losses are modeled by Equation 6. Its first two terms:

$$\frac{C_r \times FTE_a \times T_l}{2} \& \frac{C_c \times FTE_a \times T_l}{2}$$

correspond, respectively, to the ramp-up process leading to the incorporation of the newcomers and to the effort expended by the original staff coaching them. Both efforts are modeled as triangular areas.

The third term:

$$\frac{C_i \times FTE_a (FTE_a + 2 \times FTE_p - Teams) (T_b - t - T_a)}{2 \times Teams}$$

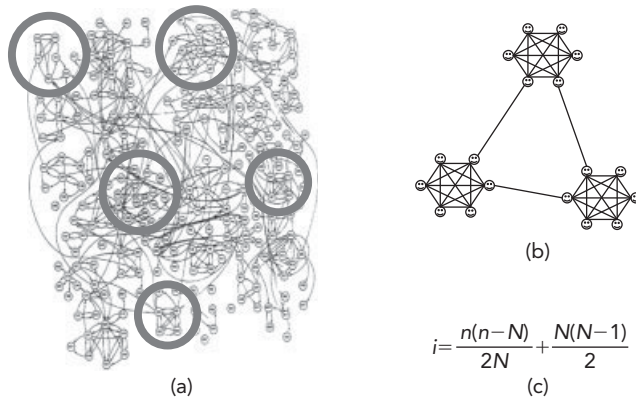
captures the extra effort expended coordinating the activities of the extended team. The equation<sup>1</sup> is derived from the findings of Thomas Allen (1984) while studying communications patterns in research and development teams (see Figure 3) and from a previous work by Miranda (2001).

## Allowing for Temporary and Permanent Staff Rates

The cost of recovery comprises the cost incurred through the overtime of the original staff and the cost, if needed, of additional resources and their overtime. Sometimes the additional staff is temporary, so for costing purposes, they might need to be considered at a different hourly rate from that of the permanent staff. The

<sup>1</sup>Notice that this equation yields a lower, but more realistic number of communication paths than the better-known  $n(n-1)/2$ .





Note. [a] Patterns of communications in R&D teams, T. Allen [1984]; [b] Stylized graph mimicking Allen's observations: everybody talks to everybody within a subsystem team while communications across subsystems are carried out by a few individuals; [c] Mathematical equation to calculate the number of communication paths.

Figure 3: Communications in R&D teams.

Recovery Cost  
( $u, t$ ) =

$$\begin{cases}
 \text{if } u \leq FTE_b[T_b + C_o(T_b - t)] \text{ then - Overtime only} \\
 \quad R_o(u - T_b FTE_b) \\
 \text{elseif } b(t)^2 - 4a(t)c(u, t) > 0 \text{ - The project can be recovered} \\
 \quad R_o C_o \times FTE_b(T_b - t) \\
 \quad + [(1 - temp)R_n + temp \times R_t] \times FTE_a(u, t)(T_b - t - T_a) \\
 \quad + [(1 - temp)R_o + temp \times R_t] \times C_o \times FTE_a(u, t) \times (T_b - t - T_a - T_i) \\
 \text{else - The project cannot be completed on time} \\
 \quad \text{Penalty} \\
 \text{endif}
 \end{cases} \quad (9)$$

$R_n$  = Normal Rate

$R_o$  = Overtime Rate

$temp$  = Proportion of temporary personnel to be employed

$R_t$  = Temporary personnel rate

$Penalty$  = Cost of not being able to complete the project on time

parameter  $temp$  in Equation 9 refers to the proportion of consultants or temporary workers employed in the project.

### Allowing for Liabilities and Opportunity Costs

Sometimes it may not be possible to recover from an underestimation within the original schedule. For example, if the decision to bring additional staff is made too close to the delivery date, the process losses incurred might be higher than the effort contributed in the time left and, as a result, the project cannot be recovered. By not being able to deliver on time, the project could incur liabilities and/or opportunity costs. These costs are accounted for in Equation 9 by the parameter  $penalty$ .

### Probabilities of Underestimation and Its Acknowledgment

Project cost and lead-time estimates based on the limited information available in tendering documents are notoriously unreliable. They are typically based on the partial results of basic design and many assumptions about its execution. The best we can do in these circumstances is to identify a range of values (see Figure 4) within which the organization believes it is possible to achieve the objectives of the project with a defined probability. The range would typically be specified through three values:

1. best-case scenario, which is the lowest amount of effort, but with a corresponding low probability of occurrence;
2. most likely scenario of some effort with the largest probability of occurring; and
3. worst-case scenario with the highest amount of effort, but again with a low probability of occurrence.

Different budgets will lead to different project approaches and different behaviors. Choosing the best-case scenario will almost certainly lead to a cost overrun and to people taking shortcuts (Austin, 2001), while choosing the worst-case scenario might

# Protecting Software Development Projects Against Underestimation

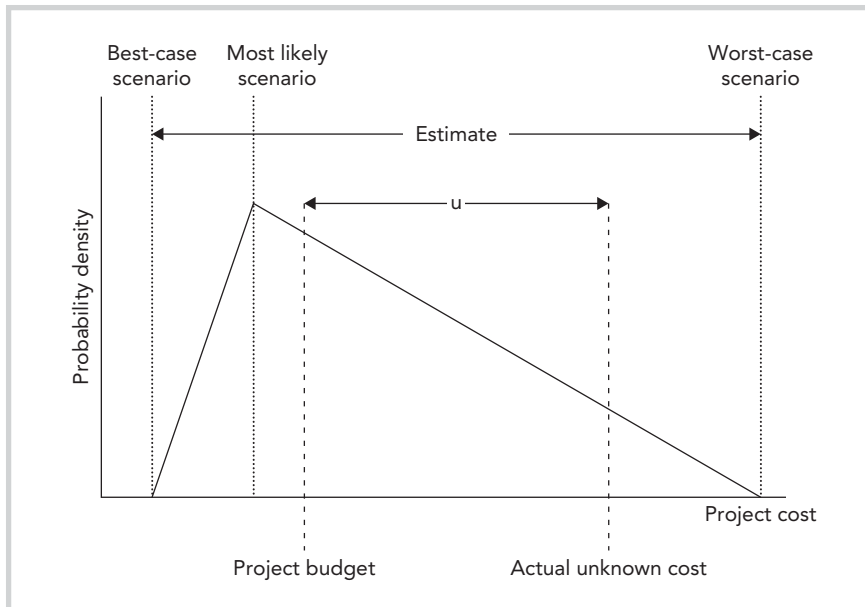


Figure 4: The probability distribution of an estimate.

result in failure to get the job and almost certain overspending (Miranda, 2003).

## Probability of Underestimation— $p(u)$

The probability distribution of the underestimation  $u$  is identical to the effort distribution in Figure 1 shifted by the project budget. The selection of a right-skewed triangular distribution is

justified for three reasons: (1) the fact that while the number of things that can go right in a project is limited and in most cases has already been factored into the estimate, the number of things that can go wrong is virtually unlimited; (2) its simplicity; and (3) since the actual distribution is not known, this choice is as sensible as any other. Equation 10 gives the cumulative probabilities for  $p(u)$ .

$$F(u) = \begin{cases} 0 & \text{if } u \leq u_{\min} \\ \frac{(u - u_{\min})^2}{(u_{\max} - u_{\min})(u_{ml} - u_{\min})} & \text{elseif } u_{\min} < u \leq u_{ml} \\ 1 - \frac{(u_{\max} - u)^2}{(u_{\max} - u_{\min})(u_{\max} - u_{ml})} & \text{elseif } u_{ml} < u < u_{\max} \\ 1 & \text{elseif } u \geq u_{\max} \end{cases}$$

$$\begin{aligned} u_{\min} &= \text{BestCaseEstimate} \\ &\quad - \text{ProjectBudget} \\ u_{ml} &= \text{MostLikelyEstimate} \\ &\quad - \text{ProjectBudget} \\ u_{\max} &= \text{WorstCaseEstimate} \\ &\quad - \text{ProjectBudget} \end{aligned} \quad (10)$$

## Probability of Acknowledging the Underestimation on a Given Month— $p(t)$

Figure 5 shows the ratio of the actual remaining duration to the current estimated remaining duration plotted as a function of relative time (the ratio of elapsed time over total actual time) for each project at each week. Under a schedule overrun condition, the estimated remaining duration will be smaller than the actual duration, and as time passes by, the estimated remaining duration will grow toward zero and the ratio will grow toward infinity. The convex pattern proves that project managers, or at least these ones, waited until the last possible minute to update the impaired schedule.

The implication of this finding for our model is that  $p(t)$  must be an increasing function of  $t$ . One such function is Equation 11 (see Figure 6).

This, of course, is not the only possibility, but it resembles the patterns in Figure 6 and it is simple. Other possibilities for the probability function

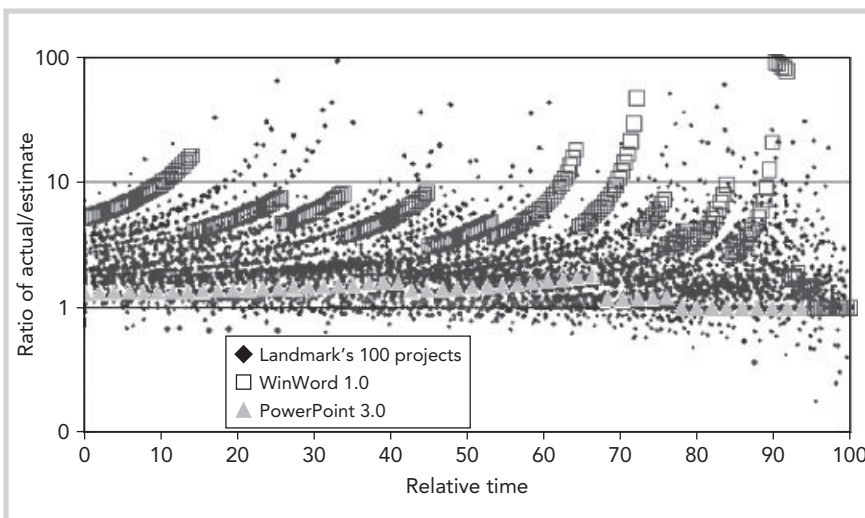
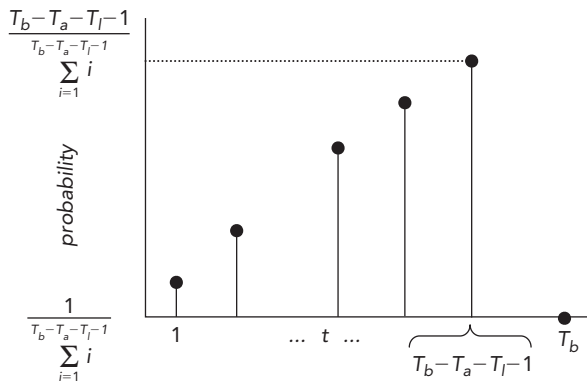


Figure 5: Procrastination as a function of project time [Little, 2006].





Note. The probability function does not extend to  $T_b$ , as the time to recruit, the time to learn, and at least one month to do some work must be taken into account.

Figure 6: Probability distribution for  $t$ .

would include the use of Bayesian probabilities to model the effect of the underestimation (e.g., larger underestimations will be easier to notice than smaller ones), but this treatment is outside the scope of the present work.

$$p(t) = \frac{t}{T_b - T_a - T_l - 1} \sum_{i=1}^t i \quad (11)$$

## Numerical Solution

Equation 1 postulates That the amount of contingency funds to budget for should be equal to the expected cost of revoery.

Although the integral could be resolved analytically, the resulting expression is complex because of the piecewise continuity of the two triangular probability distributions and the need to decompose the function in order to consider whether or not the project could be recovered through the use of overtime alone or if additional resources need to be added. Consequently, the integral will be approximated by a sum of its parts according to the following algorithm.

Figure 7 shows the contingency amounts required by each level of funding for a project with an optimistic estimate of 200 man-months,

a most likely estimate of 240 man-months, and a pessimistic one of 480 man-months with a penalty of 600 man-months. As expected, when the project is budgeted at its most optimistic estimate, the contingency is at its maximum, and when the project is budgeted at its most pessimistic level,

the contingency is zero. In the example, the minimum total cost is achieved for a budget allocation of 320 man-months. The location of the minimum would depend on the amount and the cost of overtime, temporary resources that might be employed on the recovery actions, and the penalty associated with the late delivery of the project.

## Managing the Contingency Funds

The MAIMS behavior could be explained by Parkinson's Law (Parkinson, 1958) and budget games like expending the entire budget to avoid setting precedents (Churchill, 1984; Flyvbjerg, 2005). If the MAIMS behavior is prevalent in an organization, all the budget allocated to a project will be spent irrespective of whether was needed or not, and as a consequence, there are never cost underruns, only cost overruns. This negates the basic premise that contingency usage is probabilistic. The obvious and mathematically valid solution for the effective and efficient management of the funds is to maintain them at the portfolio level, distributing

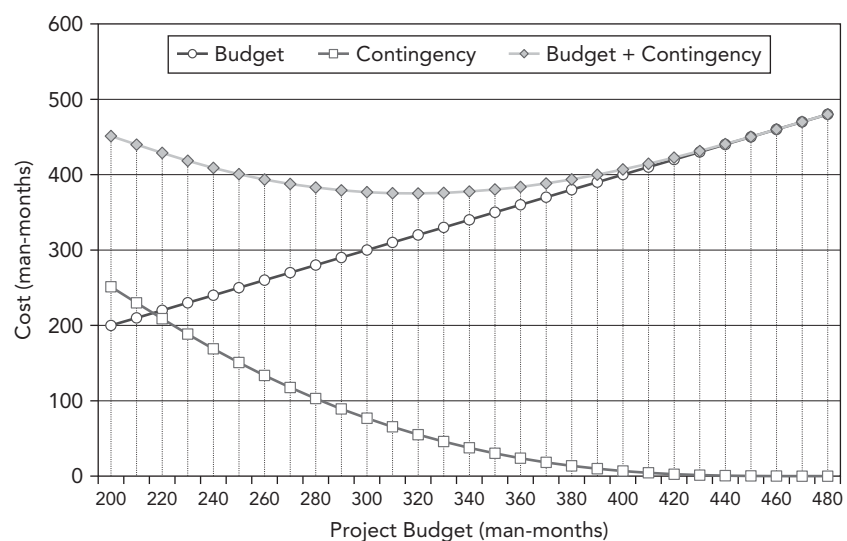


Figure 7: Breakdown of total project costs (budget + contingency) as a function of the budget allocated to the project.

# Protecting Software Development Projects Against Underestimation

them to the individual projects on an as-needed basis. This is explored in the following paragraphs by means of a Monte Carlo simulation of a portfolio consisting of three projects identical to the one in the example in Figure 7 under four different budget allocation policies.

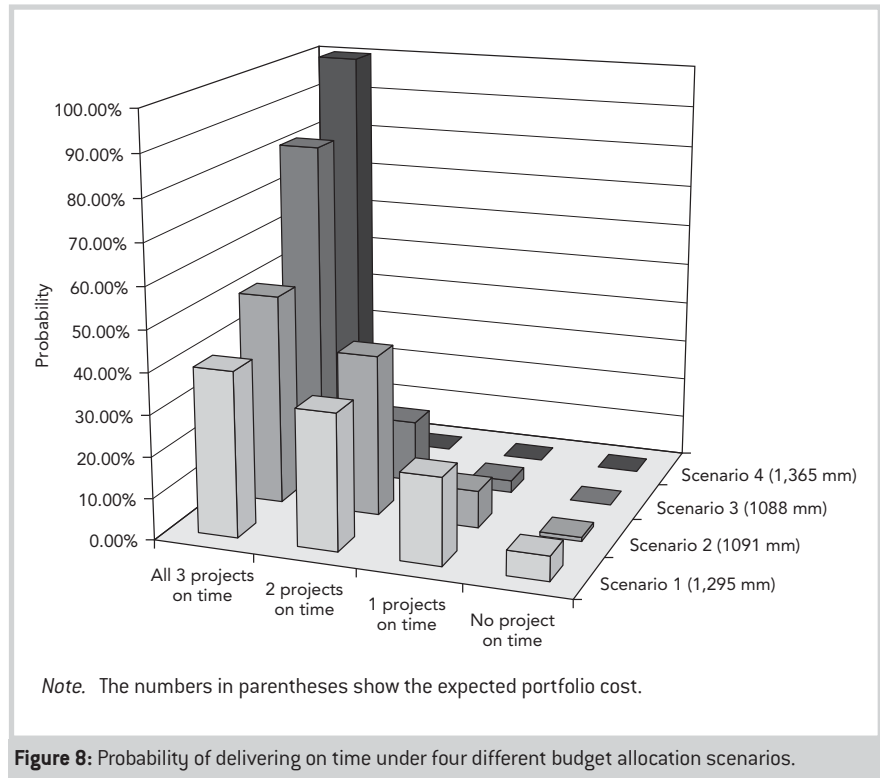
Figure 8 shows the probability of delivering on time and the expected portfolio cost for each scenario. The portfolio cost includes the allocated budget for the three projects plus their recovery costs or, whenever it is not possible to recover from the underestimation, the penalty cost.

Scenario 1 shows the result of the simulation when projects are allocated a budget equal to the most optimistic estimate (200 man-months). This is probably the worst policy of all. Not only does it yield the second-highest portfolio cost, but it also has the most late projects. Despite the projects being allocated the minimum budget, recovery costs and penalties drive the cost up.

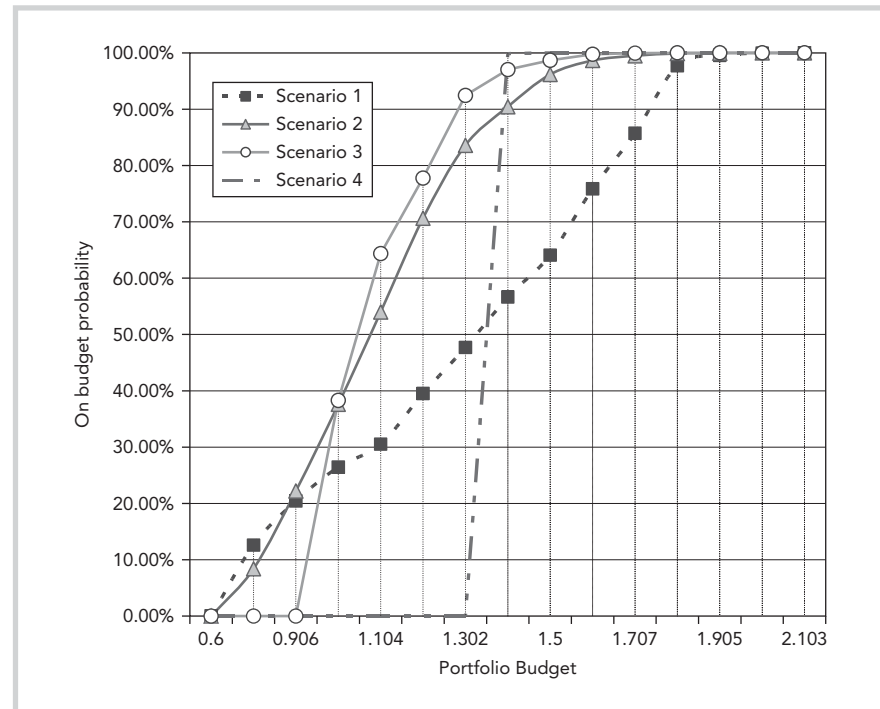
Scenario 2 corresponds to a budget allocation equal to the most likely estimate (240 man-months). In this case, the portfolio cost is lower than in the previous scenario and the probability of delivering on time is higher. Scenario 3 corresponds to a budget allocation that minimizes the expected recovery cost (contingency) as shown in Figure 7.

With a total cost of 1,088 man-months, this scenario offers the lowest expected total cost with a high probability of delivering the three projects on time. The budget allocation for Scenario 4 is set at 455 man-months, the 99% quartile of the estimate distribution. In this scenario, all projects are completed on time, but the cost is the highest.

Figure 9 shows the distribution of portfolio costs for each of the scenarios. What is important to look at here is the steepness of the curve. Steeper curves are the result of a smaller variance of the portfolio costs for a given scenario. Scenario 4 has the lowest variance since the large budgets allocated to the projects preclude underestimations. Scenario 1 is the opposite. It has the largest variance as



**Figure 8:** Probability of delivering on time under four different budget allocation scenarios.



**Figure 9:** Distribution of the portfolio costs for each scenario.

Scenario	Expected Portfolio Cost (From Simulation) (Man-Months)	Budget for the Three Projects (Man-Months)	Contingency Funds (Man-Months)	Portfolio Budget (Man-Months)	Probability of Not Exceeding the Portfolio Budget (Fig. 9)(%)
1	1,295	$3 \times 200 = 600$	$3 \times 200 = 753$	$600 + 753 = 1,353$	55%
2	1,091	$3 \times 240 = 720$	$3 \times 150 = 450$	$720 + 450 = 1,170$	68%
3	1,088	$3 \times 320 = 960$	$3 \times 55 = 165$	$960 + 165 = 1,125$	71%
4	1,365	$3 \times 455 = 1,365$	$3 \times .5 = 1.5$	$1365 + 1.5 = 1,366$	99%

**Table 1:** Summary of budgeting policies.

a result of each project being underestimated at one simulation iteration or another. The importance of the curves' steepness is that the steeper the curve, the higher the safety per dollar or man-month added to the project budget.

The results of the discussion are summarized in Table 1.

The most efficient policy is thus the one corresponding to Scenario 3, which guarantees a 71% probability of being on budget for an expected portfolio budget of 1,125 man-months. It is important to emphasize that, since we did not include, for simplicity reasons, the use of temporary personnel and overtime for extended periods in the examples, the recovery costs are not as high as they would be if it was necessary to resort to any of these two sources of additional effort.

## Summary

In this article, we postulate that a project estimate is a range of values, within which an organization believes it is possible to achieve the project's objectives with a defined probability. A budget is a political decision that results on the allocation to the project of an amount the estimated range. A low budget will have a large probability of underestimating the actual effort required. A large budget will almost certainly result in gold-plating and overengineering. As organizations tend to privilege schedule over cost, when projects are underesti-

mated management's first course of action will be to maintain the delivery date by adding resources to the project. This requires that project reflect what it would cost to recover from the underestimation and not what it would have cost to do the underestimated work had this been included from the project onset.

The proposed model takes into account the magnitude of the underestimation, the time at which the underestimation is acknowledged, and the consequences of not delivering on time, and can be used to either calculate contingencies in actual projects or for education purposes.

## Acknowledgments

We would like to thank Gaetano Lombardi from Ericsson for his valuable comments. This research project has been funded partially by the European Community's Sixth Framework Program—Marie Curie International Incoming Fellowship under contract MIF1-CT-2006-039212. ■

## References

- Allen, T. (1984). *Managing the flow of technology*. Cambridge, MA: MIT Press.
- Austin, R. (2001). *The effects of time pressure on quality in software development: An agency model*. Boston: Harvard Business School.

Babad, E., & Katz, Y. (1991). Wishful thinking: Against all odds. *Journal of Applied Social Psychology*, 21, 1921–1938.

Brooks, F. (1995). *The mythical man-month: Essays on software engineering*. Indianapolis, IN: Addison-Wesley.

Churchill, N. (1984, July-August). Budget choice: Planning vs. control. *Harvard Business Review*, pp. 150–164.

Flyvbjerg, B. (2005). Design by deception: The politics of megaprojects approval. *Harvard Design Magazine*, 22, 50–59.

Gordon, C. (1997). Risk analysis and cost management (RACM): A cost/schedule management approach using statistical cost control (SCC). Retrieved July 9, 2008, from [http://www.geocities.com/mtarrani/RiskAnalysis\\_and\\_CostManagementOverview.pdf](http://www.geocities.com/mtarrani/RiskAnalysis_and_CostManagementOverview.pdf)

Grey, S. (1995). *Practical risk assessment for project management*. New York: Wiley.

Kujawski, E., Alvaro, M., & Edwards, W. (2004). Incorporating psychological influences in probabilistic cost analysis. *Systems Engineering*, 3(7), 195–216.

Little, T. (2006). Schedule estimation and uncertainty surrounding the cone of uncertainty. *IEEE Software*, 23(3), 48–54.

Miranda, E. (2001). *Project screening: How to say "no" without hurting your career or your company*. Paper

## Protecting Software Development Projects Against Underestimation

presented at the European Software Control and Measurement Conference, London, England.

**Miranda, E. (2003).** *Running the successful high-tech project office*. Boston: Artech House.

**Parkinson, C. (1958, November).** *Parkinson's law: The pursuit of progress*. London: John Murray.

**Project Management Institute (PMI). (2004).** *A guide to the project management body of knowledge (PMBOK® guide)* (3rd ed.). Newtown Square, PA: Author.

**Sim, S., & Holt, R. (1998).** The ramp-up problem in software projects: A case study of how software immigrants naturalize. *Proceedings of the 1998 International Conference on Software Engineering* (pp. 361–370).

**Tykocinski, O., Pittman, T., & Tuttle, E. (1995).** Inaction inertia: Foregoing future benefits as a result of an initial

failure to act. *Journal of Personality and Social Psychology*, 68, 793–803.

---

**Eduardo Miranda** is an independent consultant and a PhD student at the École de Technologie Supérieure, Université du Québec (Montréal, Canada), where he also teaches project management courses to graduate students in software engineering. He has 20 years of experience in the management and development of information systems and R&D projects in the information systems, aerospace, and telecommunications industries. He holds a master's degree in project management from the University of Linköping, Sweden, a master of engineering degree from the University of Ottawa, Canada, and a bachelor of science from the University of Buenos Aires, Argentina. He has published over 10 papers on software development methodologies, estimation, and project management and is the author of the book *Running the Successful Hi-Tech Project Office* published in March 2003.

---

**Alain Abran**, a native Canadian, holds a PhD in electrical and computer engineering (1994) from École Polytechnique de Montréal (Canada) and master's degrees in management sciences (1974) and electrical engineering (1975) from the University of Ottawa. He is a professor and the director of the Software Engineering Research Laboratory at the École de Technologie Supérieure, Université du Québec (Montréal, Canada). He has 15 years of experience in teaching in a university environment as well as more than 20 years of industry experience in information systems development and software engineering. His research interests include software productivity and estimation models, software engineering foundations, software quality, software functional size measurement, software risk management, and software maintenance management. He is currently a member and coeditor of the Guide to the Software Engineering Body of Knowledge project and a cochair of the Common Software Measurement International Consortium (COSMIC).

---

## Appendix: Model Parameters

$C_i$  = Effort expended attending to one interaction (%)

$C_o$  = Maximum overtime to be employed (%)

$C_r$  = Effort to be expended in preparation for the arrival of the newcomers (%)

$C_c$  = Effort expended in coaching a newcomer (%)

$C_d$  = Decline in performance due to fatigue (%)

$FTE_b$  = Budgeted full-time equivalents

$lag$  = Time after which overtime productivity starts to decline due to fatigue

$T_a$  = Average time for newcomers to arrive

$T_l$  = Average time for newcomers to get up to speed

$T_b$  = Budgeted project duration

$Teams$  = The number of teams (subsystems) into which the project is organized

$Penalty$  = The amount to be used as cost of recovery when the project cannot deliver on time

## **ANNEX III**

### **Agile Monitoring Using the Line of Balance**



Contents lists available at ScienceDirect

## The Journal of Systems and Software

journal homepage: [www.elsevier.com/locate/jss](http://www.elsevier.com/locate/jss)

## Agile monitoring using the line of balance

Eduardo Miranda<sup>a,\*</sup>, Pierre Bourque<sup>b</sup><sup>a</sup> Institute for Software Research, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213, United States<sup>b</sup> École de Technologie Supérieure, Université du Québec, Canada

## ARTICLE INFO

## Article history:

Received 28 July 2009

Received in revised form 31 December 2009

Accepted 24 January 2010

Available online xxxx

## Keywords:

Scrum

Feature Driven Development

Project management

Tracking and control

Line of balance

Release planning

Burn down charts

Cumulative flow diagrams

Agile methodologies

LOB

## ABSTRACT

There is a need to collect, measure, and present progress information in all projects, and Agile projects are no exception. In this article, the authors show how the line of balance, a relatively obscure indicator, can be used to gain insights into the progress of projects not provided by burn down charts or cumulative flow diagrams, two of the most common indicators used to track and report progress in Agile projects. The authors also propose to replace the original plan-based control point lead-time calculations with dynamic information extracted from a version control system and introduce the concept of the ideal plan to measure progress relative to both, end of iteration milestones and project completion date.

© 2010 Elsevier Inc. All rights reserved.

## 1. Introduction

Progress monitoring and reporting is a basic function during the execution of any project. Progress needs to be monitored so potential problems can be adverted before they materialize. Besides being required to steer the project, timely and accurate reporting is essential to keep stakeholder support and funds flowing.

With the exception of earned value reporting (Project Management Institute, 2004), which is almost mandatory in most large government-sponsored projects (GAO, 2007), few, if any, tracking and reporting mechanisms have been standardized. Despite this, certain practices have emerged as preferred within some Agile communities and not in others. For example, burn down charts (Schwaber and Beedle, 2004) are favored by the Scrum community, cumulative flow diagrams (Anderson, 2004a,b; Microsoft, 2006) by feature driven development (FDD) practitioners, and stories completed and tests passed (Wake, 2001) by Xp adepts.

While simple to produce and easy to understand, these charts do not communicate the whole picture. Burn down charts, stories completed and tests passed report how much work is left and provide some indication of where the project ought to be, had it progressed at a constant rate. None of them report work in progress. Cumulative flow diagrams on the other hand, report work in

progress but fail to relate it to how much should have been accomplished if the project is to meet its commitments.

The reason why it is important to consider work in progress vs. any commitments made, is that this information allows the team to focus its resources where they are needed the most. For example, should the work in progress indicators point to a bottleneck in the testing activities the team could redirect its efforts from coding to testing. By helping balance the different activities in the production chain, the added visibility allows the team to deliver at its maximum velocity. Reporting work in progress also helps communicate to outside stakeholders that the team is advancing, even if user stories are not being completed daily.

In this paper, we propose the use of the line of balance (LOB) (Office of Naval Material, 1962) method as an alternative to overcome the deficiencies noted above. In keeping with the idea of using the team's actual performance, we also propose to derive the lead-times for the so called "control points", not from an activity network as in the original LOB formulation, but from the team's velocity.

To illustrate the concepts presented, we have chosen artifacts and processes from Scrum (Schwaber and Beedle, 2004) and FDD (Coad et al., 1999) as examples. The reader, however, should have no problem extending them to other contexts.

Section 2 discusses current approaches to progress monitoring and reporting in Agile projects. In Section 3 we describe the LOB method and we then pursue to explain how LOB outputs are

\* Corresponding author.

E-mail address: [mirandae@andrew.cmu.edu](mailto:mirandae@andrew.cmu.edu) (E. Miranda).



interpreted, Section 4, dealing with changes, Section 5, the extension of LOB methods to teams of teams, Section 6, extending the LOB to portfolio management, Section 7 and implementation of the method, Section 8.

## 2. Progress monitoring and reporting in Agile projects

Agile methods are characterized by the recurring end-to-end development of discrete software system capabilities. That is, instead of evolving the whole software, or large chunks of it, through the stages of the development life cycle (Fig. 1a), following a brief up-front analysis phase, they break down the total effort into small self-contained pieces of value called user stories or features; and each of them is evolved through the entire life cycle, and with the exception of technical dependencies, mostly independently of the others. The sequence design build test integrate (DBTI) is repeated over the life of the project as many times as user stories are, generating partial versions of the complete software system along the way (Fig. 1b).

In a project using Scrum, progress is tracked and reported by means of a release burn down chart, an iteration burn down chart and a task board (Cohn, 2006). The charts are called “burn down” because they show what work remains to be done rather than what work has been completed.

The release burn down chart (Fig. 2a) is used to monitor and report the overall progress of the project to both sponsors and team members. The release burn down chart shows two key indicators: the overall rate of progress and the amount of work remaining. By extrapolating the rate of progress, it is possible to forecast the time of completion. If work is added to the project, the curve is adjusted upwards, if work is dropped it is adjusted downwards.

Iteration burns down charts (Fig. 2b) are derived from the task board information (Fig. 3), and its audience is the team members. The purpose of the chart is to show the number of hours of work left vs. the number of days left on the current iteration. The devel-

opment team uses this information to conclude whether all the work of the iteration can be completed at the current pace or whether a couple of extra hours would be needed or some work would have to be rescheduled.

The task board adds a great deal of information by showing the breakdown of a user story into tasks. Tasks are embodied in task cards. At a minimum, the task cards identify the type of task, e.g. coding, writing test cases, integrating, etc. and the number of hours estimated for its execution. As work progresses, team members move task cards from one state (pending, assigned, in progress, completed) to another. The board provides the means to coordinate work among team members and the raw data, i.e. hours of work left, to produce the iteration burn down chart. Other information on the board, such as how many user stories are being coded or how many are being tested, is not exploited – at least in a structured way, under the assumption that all that counts is work completed and work remaining.

The cumulative flow chart (Jackson, 1991; Anderson, 2004a,b), is constructed by counting the number of user stories that have reached a certain state of development at a given time. Compared to the burn down chart, the cumulative flow diagram favored by FDD practitioners, offers a wealth of information: rate of flow, quantity in process and time in process (Fig. 4). Unlike the line of balance status chart, to be presented later, cumulative flow diagrams do not show target information.

Of special interest is the use of earned value techniques in Agile projects which has been more in response to reporting requirements (Alleman et al., 2003; Rusk, 2009) than a choice of the development teams and which has required some tweaking or reinterpretation of some fundamental concepts to be applicable (Cabri and Griffiths, 2006; Sulaiman et al., 2006). When used at the project level, earned value is more a reporting mechanism between the project sponsor and the team doing the work than a diagnostic tool. It lacks the visibility required to take any decision. To be used as a diagnostic tool, earned value requires the identifi-

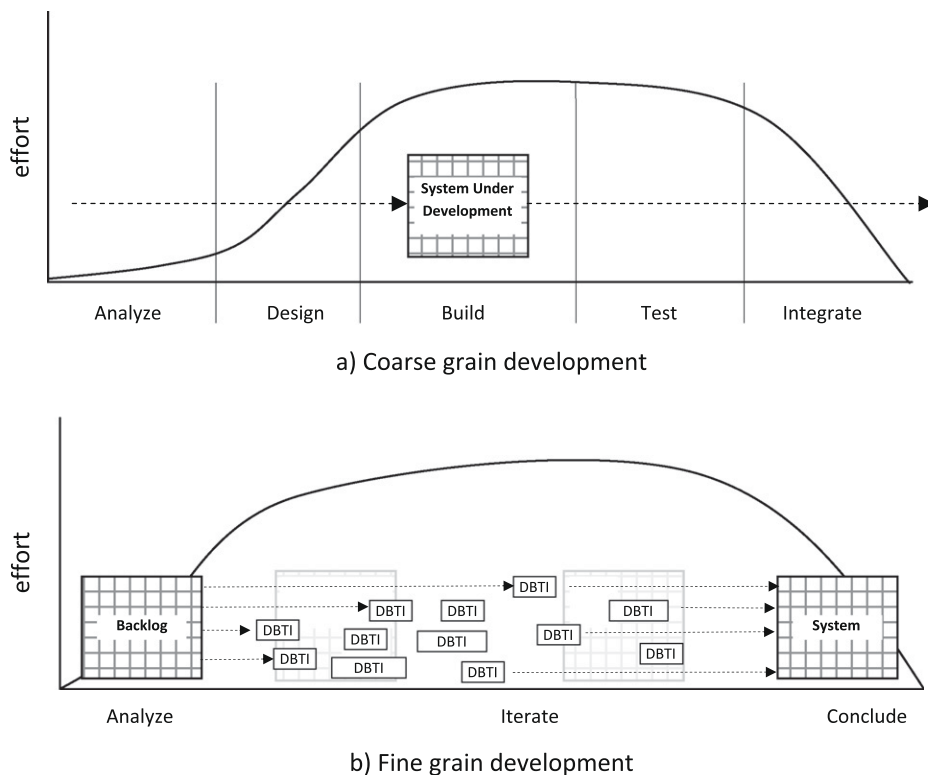


Fig. 1. Software life cycles: (a) coarse grain development, and (b) fine grain development – lightly shaded squares represent partial versions of the system.



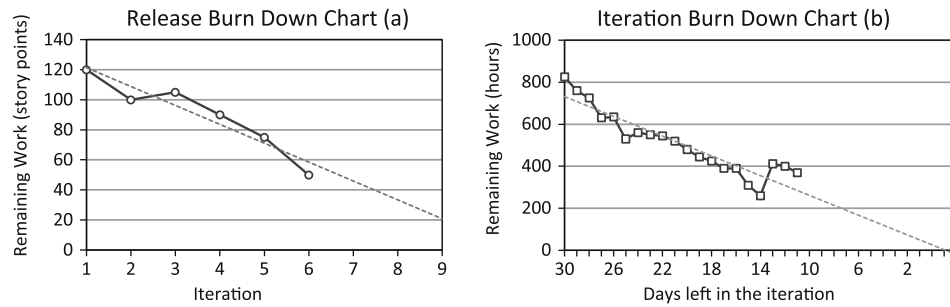


Fig. 2. Burn down charts: (a) release chart; (b) iteration chart.

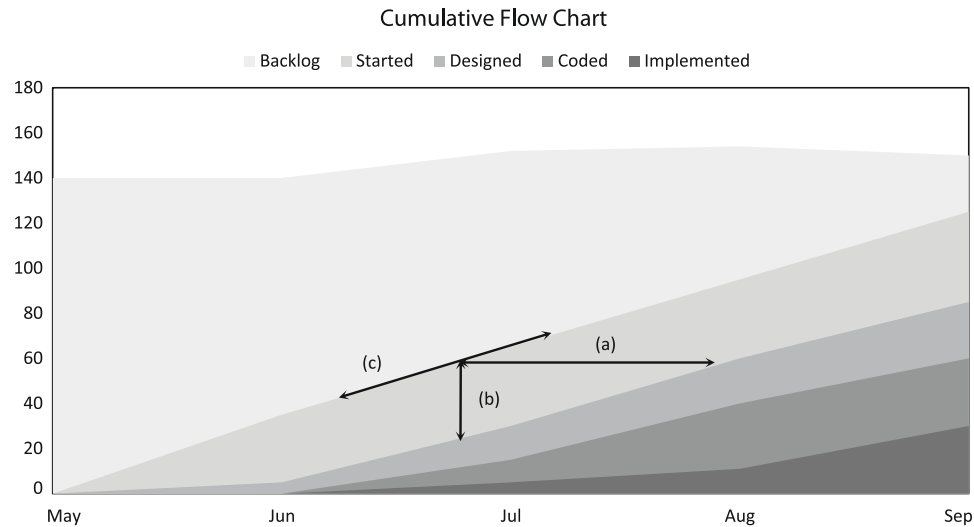
As of 5/28/08		Iteration ends 6/12/08		Work days left: 11	
User story	To Do (1)	Assigned (2)	In Progress (3)	Completed	Hours Left (1+2+3)
User story 1	Integrate	Test	Code	Design	28
User story 2		Integrate	Test	Code Design	8
• • •	...	...	...	...	298
User story n	Code Design Test Integrate				36
				Total Hours Left	370

Fig. 3. Scrum task board showing the status of all the tasks included in the iteration.

cation of deliverables and their contributing tasks by means of a WBS or similar arrangement, the definition of their start and end dates, the allocation of a budget, and a time reporting system capable of tracking data at the same level of granularity. Because these conditions are rarely found in Agile projects its usefulness is limited to that of a burn down chart with the addition of spending reporting.

### 3. The line of balance method

The purpose of the LOB method is to ensure that the many activities of a repetitive production process stay “in balance” that is, they are producing at a pace which allows an even flow of the items produced through a process and at a speed compatible with the goals set forth in a plan. The method does this by calculating



**Fig. 4.** Cumulative flow chart the upper series (backlog) shows the total amount of work to be done. The ups and down correspond to work added or dropped, respectively. The horizontal line (a) measures the average time in state for each user story. The vertical line (b) reports the number of user stories in a given state at a particular time. The inclined line (c) represents the rate at which user stories reach a particular state.

how many items should have passed through a given operation or control point, and showing these figures alongside the number that actually did (Al Sarraj, 1990; Arditi et al., 2002) (Fig. 5). In the context of Scrum an item would be user story and in the case of FDD, a feature.

The LOB method was devised by the members of a group headed by George E. Fouch during the 1940s to monitor production at the Goodyear Tire & Rubber Company, and it was also successfully applied to the production planning of the huge US Navy mobilization program of World War II and during the Korean hostilities. Today, the LOB method is applied to a wide spectrum of scheduling activities, including research and development, construction flow planning, and tracking the progress of responses to trouble reports (Miranda, 2006; Harroff, 2008).

The LOB Status Chart in Fig. 5 shows the project progress as of December 28th. The plan for the project is to deliver a total of 123 user stories. This is shown by the control point labeled "Backlog", which shows the total amount of work the team is currently

committed to deliver. For the other control points, the chart displays two columns: the "Planned" column showing how many items should have passed through the control point according to the proposed production or release plan and the "Actual" column showing how many did actually pass through it.

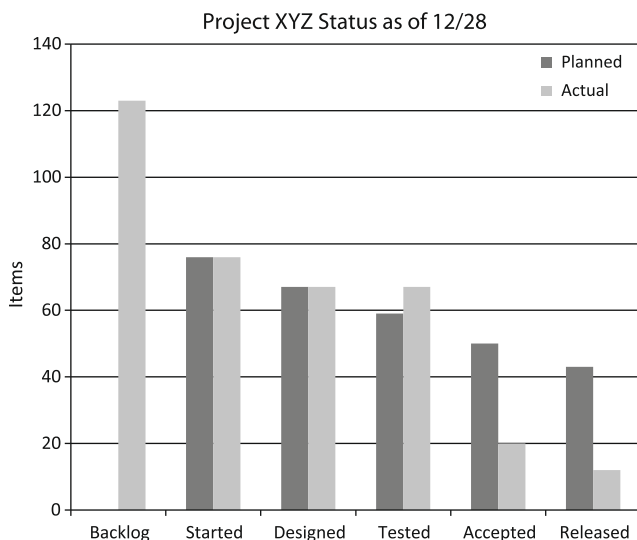
By comparing the number of planned items to the number of actual items, we can see, for example, that the activities leading to the "Started" and "Designed" control points are on track with respect to the delivery plan, and that the testing activities are a little ahead of schedule. In contrast, the activities leading to the "Accepted" and "Released" control points are behind schedule. According to the plan, there should have been around 50 user stories accepted by this time, but in fact there are only 20, and, since the testing activities are ahead of schedule, the problem must lie with the activities leading to acceptance. The chart does not show the cause of the problem; however it is clear that whatever the reason, the slow pace of the acceptance activities is jeopardizing the next release.

The advantages of the LOB method over burn down charts and cumulative flow diagrams are that the LOB:

- o Shows not only what has been achieved, but also what was supposed to be achieved in a single chart.
- o Shows work in progress, permitting a more accurate assessment of the project status.
- o Exposes process bottlenecks, allowing the team and the project sponsor to focus on the points causing slippages.

Knowing how many user stories are in a given state allows us to answer the question: Where are we today? But leaves unanswered the more fundamental one of: Where are we in relation to where we planned to be? To answer this question the LOB method identifies (College, 2001):

- o A number of control points at which progress is to be monitored.
- o A delivery plan specifying the number of user stories to be produced in each iteration.
- o A status chart, showing the number of user stories that have passed through each control point vs. the number that should have passed through according to the delivery plan.



**Fig. 5.** A line of balance status chart showing the number of items that should have passed through a given control points vs. how many actually did.

### 3.1. Control points

In LOB terminology, a control point is a point in the development process with a well defined exit criterion at which work in progress and work completed are measured. In the context of tracking user stories, control points would correspond to all or some of the states comprising the user story's life cycle (Fig. 6). Knowing the status of the project at any given time requires knowing the state of each and every user story.

A control point's lead-time (Fig. 7) is the average time it takes a user story to move from that point to the point at which the user story is considered completed. In the original LOB method, these times are derived from an activity network comprising the activities involved in producing a user story while in our proposal they are calculated by measuring the average time a user story spends in each state (Fig. 8).

The time each user story spends in each state is calculated using expression (1). The average time in a state for a given state could be calculated as either the median, the arithmetic mean or the mode of the individual times in state (2). The data required by these calculations is readily available from most version control systems.

$TimeInState_{qi}$

$$= \begin{cases} \text{if } TransitionDate_{(q+1)i} \text{ exists then} \\ TransitionDate_{(q+1)i} - TransitionDate_{qi} \\ \text{otherwise} \\ CurrentDate - TransitionDate_{qi} \end{cases} \quad (1)$$

$q = 1, 2, \dots, n$  is a control point mapped to one of the user story's lifecycle states.

$i = 1, 2, \dots, v$  is used to denote an individual user story.

$\overline{TimeInState}_q$

$$= \begin{cases} Median(TimeInState_{q1}, TimeInState_{q2}, \dots, TimeInState_{qv}) \\ \text{or} \\ ArithmeticMean(TimeInState_{q1}, TimeInState_{q2}, \dots, TimeInState_{qv}) \\ \text{or} \\ Mode(TimeInState_{q1}, TimeInState_{q2}, \dots, TimeInState_{qv}) \end{cases} \quad (2)$$

The median is preferred over the arithmetic mean to prevent rare but very complex, or very simple, user stories from skewing the value of the statistic. This can be observed, for example, in Fig. 8 by noting that the arithmetic mean for "Accepted" lies outside its interquartile range, meaning that, while most user stories were accepted in one-day, a few of them took longer, driving its value up to 2.31 days. The use of this value in the calculations instead of the far more common one-day median, will inflate the number of user stories reported in the planned column of the "Accepted" control point signaling that a higher number of them should pass through it than they actually needed to.

Moving from times in a state to lead-times (3) is straightforward. Since the mean time spent in the "Accepted" state (Fig. 7) is the typical time it takes a user story to go from the "Accepted" to the "Released" control point, the lead-time for "Accepted" is 1 day. In the case of the lead-time for the "Tested" control point, a user story typically spends 5 days in the "Tested" state and then 1 more in the "Accepted" state. This is equivalent to saying that it takes 5 days to move from "Tested" to "Accepted" plus 1 day in moving from "Accepted" to "Released". Consequently, the lead-time for the "Tested" control point is 6 days. Table 1 shows the lead-time calculations for the control points in Fig. 6.

$LeadTime_n = 0$

$$LeadTime_{q \in n-1, n-2, \dots, 1} = LeadTime_{q+1} + \overline{TimeInState}_q \quad (3)$$

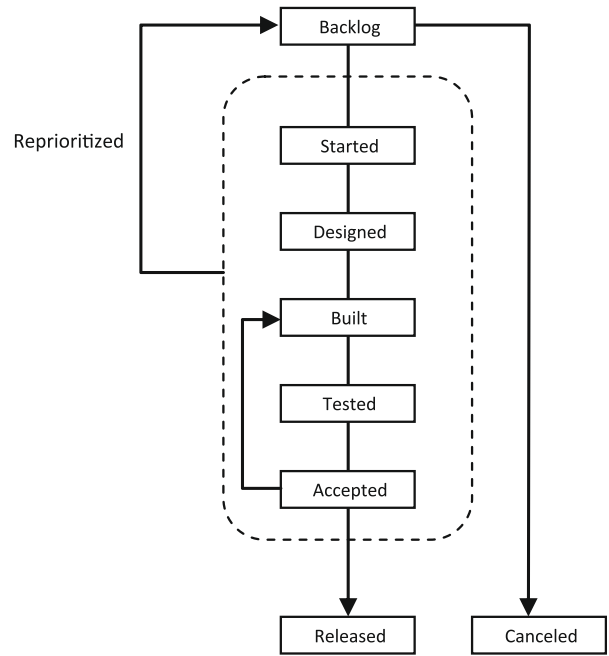


Fig. 6. A user story's typical life cycle.

Although different user stories will require different efforts to implement them, we will treat them as being of equal size, as most teams strive to achieve this goal when breaking down the total effort, and that it is very unlikely that in planning an iteration, a team will include all the large stories in one and all the small ones in another. Should the assumption of equal size prove inappropriate, user stories should be normalized, i.e. reduced to a common denominator, using story points, function points, man-hours, or any other measure that accounts for the relative development effort required by them.

### 3.2. The delivery plan

The delivery plan (Fig. 9) comprises two sub plans, the Release plan (RP) and the Ideal Plan (IP). The RP specifies how much capa-

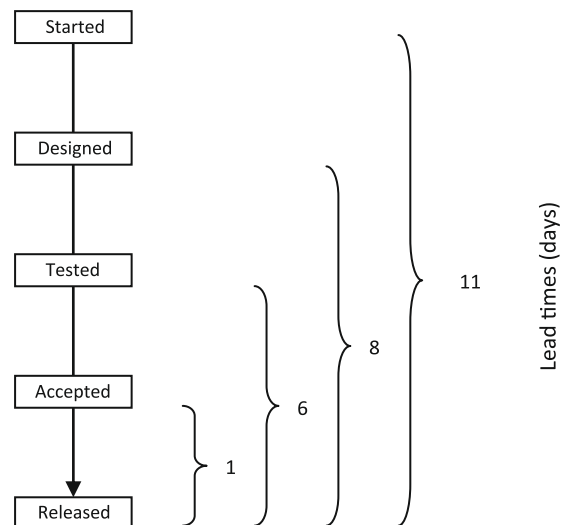
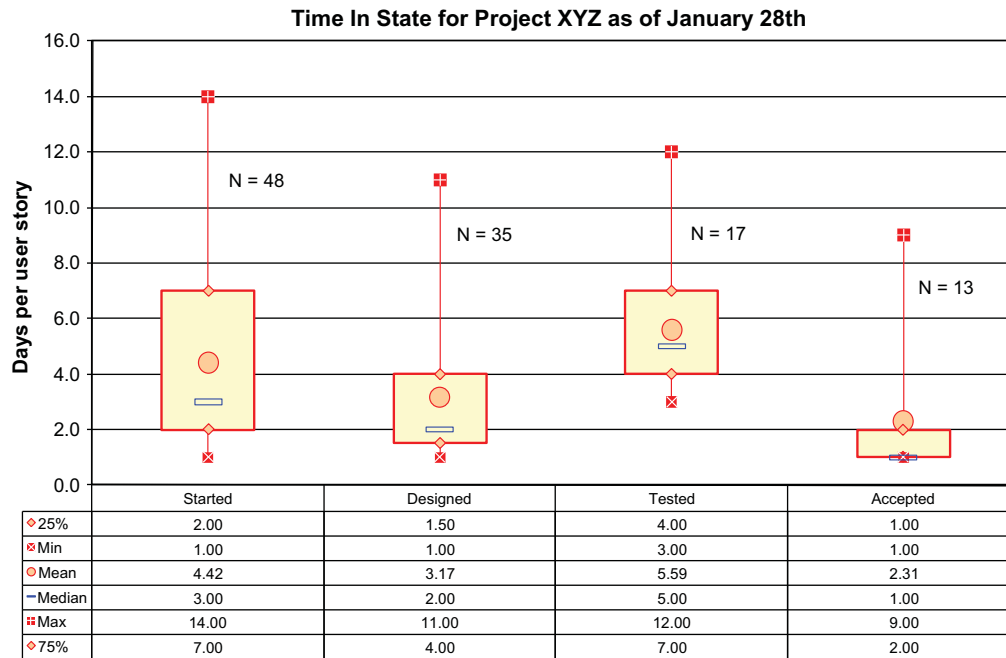


Fig. 7. Control points and lead-times. Note that not all the states mentioned in the user story's life cycle have been included for control. The decision to do this was based on the amount of visibility desired or requested. After the user story is released, it is no longer tracked.



**Fig. 8.** Box-plot chart showing the distribution of times spent in each state by the user stories developed so far. The 25 and 75% quartiles delimit the range within which the middle 50% of the values are included. *N* denotes the number of user stories included in the statistic.

**Table 1**

Lead-time calculations.

Control point	Time in a state (days)	Lead-time (days)
Released		0
Accepted	1	0 + 1 = 1
Tested	5	1 + 5 = 6
Designed	2	6 + 2 = 8
Started	3	8 + 3 = 11

bility will be delivered at the end of each iteration, as agreed between the team and the project sponsor while the *IP* shows the proportion of capability that should have been delivered at any given time, assuming constant work progress throughout the project.

The *RP* is prepared by the project team based on its own estimation, the team's velocity and their resource availability. The plan can be adjusted later, with the agreement of the sponsor, to reflect the team's actual performance and changes to the product backlog. The *RP* in Fig. 9 shows that the software system to be delivered consists of 150 user stories that must be delivered by March. Based on their previous experience and business needs, the team breaks down the total delivery into five releases. The first release, due by the beginning of November, consists of 25 user stories; the second release, due by the beginning of December, includes 35 user stories, the increase in velocity accounts for the learning effects, i.e. the team becoming more proficient with the application. In December, because of the holiday period, the team's availability is reduced and the team only commits to the delivery of 15 user stories. For the last two months, the team expects to bring some additional experienced resources on board, which will help them increase the delivery rate to 35 and 40 user stories, respectively.

The *IP* is built by joining the start and the expected end of the project with a straight line. The slope of the line is the average productivity that the team will need to exhibit to deliver what has been agreed with the project sponsor in the time the team has committed to. The *IP* helps keep the plan honest by raising the following questions: Is the average productivity reasonable, e.g. has it been observed before in like projects? Are there any periods in

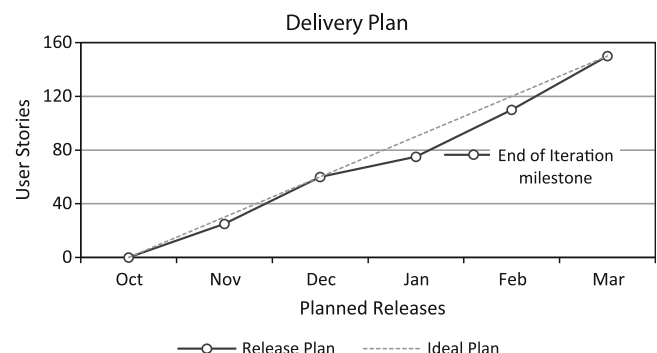
which the team is counting on heroic productivity levels? Are there any periods in which productivity is well below par? Why?

While the *RP* plan enables tracking progress against current and past iterations, the *IP* facilitates the assessment of delays against the total project schedule.

### 3.3. The status chart

In the original formulation of the LOB, the status chart (SC) only provided quantitative information about the work in progress and the work completed relative to the end of iteration milestones in the *RP*. To these, the authors added a third indicator showing the progress to be achieved relative to the *IP*, in order to provide a strategic view that could prevent overreactions to missed deadlines and provide early warnings to unfavorable trends. The use of this indicator will be exemplified in the section of interpreting the status chart.

To calculate the number of user stories that should have been ready at a time *t* relative to the *RP*, we need first to mathematically



**Fig. 9.** Delivery plan proposed by the development team. The *RP* curve is used to calculate progress relative to the end of the iterations while the *IP* provides a baseline against which to compare the promised deliveries to an average productivity over the life of the project.

express it (4) as a series of straight lines, each valid in the  $[t_i, t_{i+1}]$  range.

$$RP_t = \begin{cases} a_1 + b_1 t & t_0 \leq t < t_1 \\ a_2 + b_2 t & t_1 \leq t < t_2 \\ \vdots & \\ a_r + b_r t & t_{r-1} \leq t < t_r \end{cases} \quad (4)$$

$$b_{i=1,2,\dots,r} = \frac{UserStories_i - UserStories_{i-1}}{t_i - t_{i-1}}$$

$$a_{i=1,2,\dots,r} = UserStories_{i-1}$$

$r$  = number of planned releases

Similarly, to calculate the planned quantities with respect to the  $IP$  we need first to find the Eq. (5) of the line that passes through  $\{(0, 0), (t_r, UserStories_r)\}$ .

$$IP_t = bt$$

$$b = \frac{UserStories_r}{t_r} \quad (5)$$

To calculate  $RP_{t_q}$  or  $IP_{t_q}$ , the number of user stories that should have passed through control point  $q$  at time  $t$ , we simply look ahead by  $q$ 's lead-time (6) and apply either (4) or (5).

$$t_q = LeadTime_q + t \quad (6)$$

The idea behind the procedure is simple. If it takes 11 days on average for a user story to go through its complete development cycle, on any given day we should have started at least as many

user stories as we are supposed to be delivering 11 days from now. Fig. 10 exemplifies this. The same idea applies to any other control point.

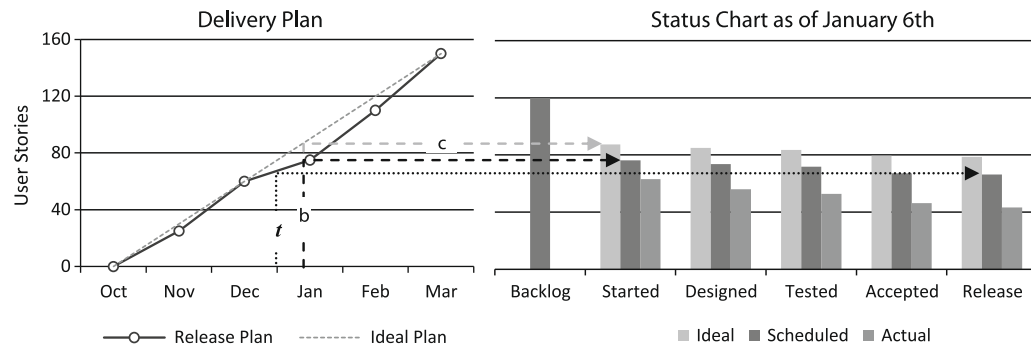
#### 4. Interpreting the status chart

In this section, we discuss three examples and give some advice on what to look for in the status chart. The examples have been purposely designed to highlight the characteristic that we want to show.

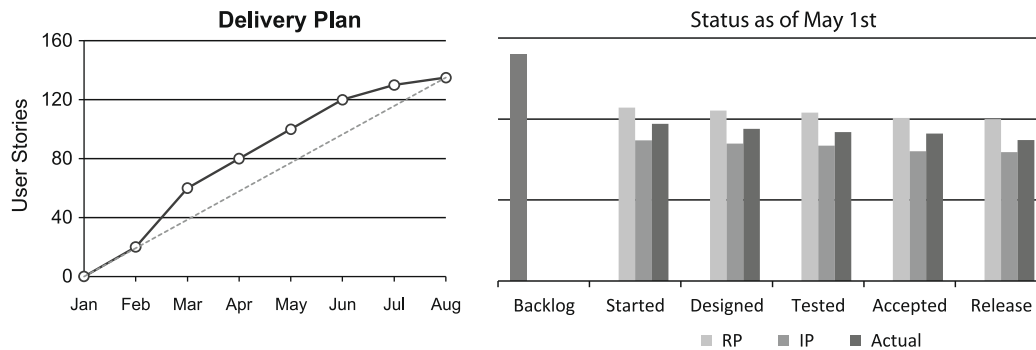
The example in Fig. 11 shows a project with a very aggressive delivery – twice the amount of user stories – targeted for March. Note the upward deviation of the  $RP$  line from the  $IP$  line. The status chart shows that all activities are in balance, since there are no abrupt falls from one state to the other. The chart also shows that the project is slightly behind with respect to its  $RP$ . They were probably too optimistic about their ability to deliver in March, but a little bit ahead of schedule with respect to the  $IP$  plan. If the team can keep the pace, it is probable that they will finish on time.

The chart in Fig. 12 shows the same project as in Fig. 11, but this time the project is well behind with respect to both the  $RP$  and the  $IP$ . Note that there are as many user stories designed as started, but there are half as many that have gone through the “Tested” control point, which points to a bottleneck in the testing activities that seems to be starving the rest of the production chain. Should the team decide to act on this information, they would need to focus their effort on testing.

In the last example (Fig. 13), the team has been very cautious and provided some allowance for the team to climb the learning



**Fig. 10.** The intersection between the time now ( $t$  = January, 6th) line and the release plan yields the  $RP_{Released}$  value, that is, the number of user stories that should have passed through the “Released” control point as of that date, to meet the target for the 4th release. The intersection between the release plan and the line (b) at  $t + LeadTime_{Started}$  yields the  $RP_{Started}$  value for the “Started” control point. The intersection of the line (c) at  $t + LeadTime_{Started}$  with the ideal plan yields the  $IP_{Started}$ .



**Fig. 11.** The team has committed to a very ambitious target for the second release. Note that, except for the month of March when the team proposes to deliver 40 user stories, the velocity is estimated at 20 or less user stories per month.

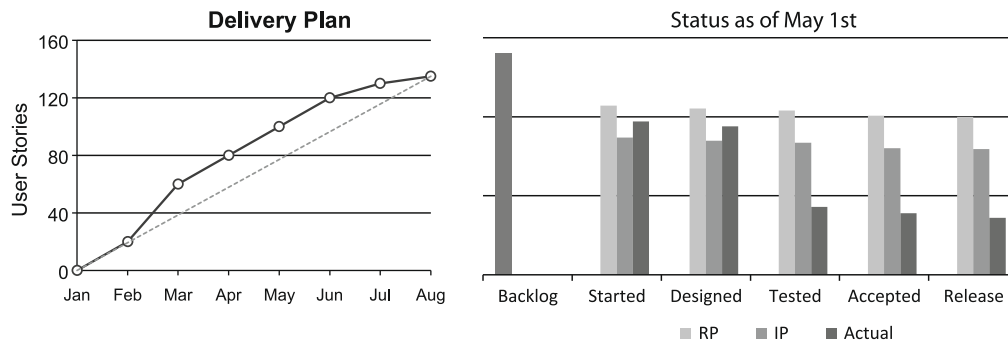


Fig. 12. Chart showing a problem with testing.

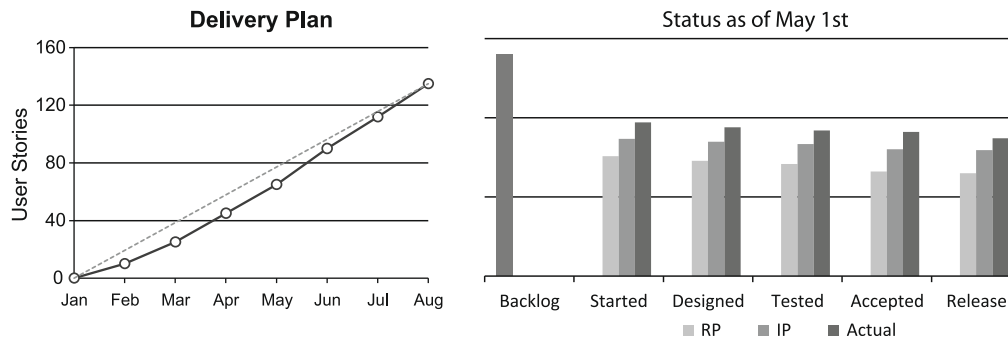


Fig. 13. This team has included an allowance for learning in their release planning. Note how the estimated velocity increases towards the end.

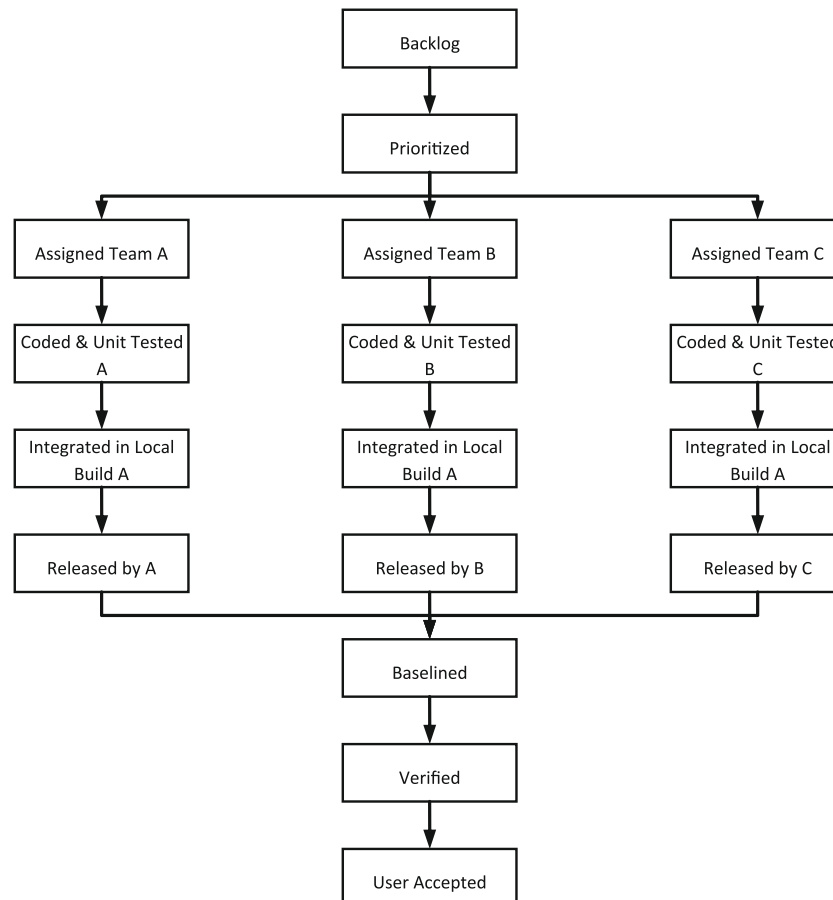


Fig. 14. Control points for a project to be executed using a team of teams.

curve. Note the concave shape of the *RP* curve. According to the Status Chart, the project is ahead of schedule with respect to both the *RP* and the *IP*.

### 5. Dealing with changes in scope

So far we have not discussed what happens when user stories are added, changed or dropped from the backlog, which as every experienced developer knows, is a constant in all projects. Adding new user stories or changing already developed ones, pose no problem. For each new or changed user story we will just add one, or in its defect, the corresponding number of user points to the backlog. If there were a need to distinguish between what has been initially agreed and any subsequent changes in the scope of the project, a “Baseline” column could be displayed beside the “Backlog” column to highlight the requirements churn.

For abandoned user stories, one must distinguish between those that have been started and those which have not. In the later case we will just subtract the corresponding quantity from the

backlog while in the case where the user story is abandoned after being started, one would have to subtract the corresponding quantity from the backlog and all the columns to which it was added to prevent the LOB chart from reporting progress on things that are not longer desired. This could be easily done, since we know what control points the user story went through.

Simultaneously with this the delivery plan would need to be adjusted to reflect the new workload.

### 6. The LOB in large Agile projects

Most Agile projects are organized around small teams of 7–15 people, with larger teams organized in hierarchical fashion as teams of teams. An example of this type of arrangement is the Scrum of Scrums organization proposed in (Schwaber and Beedle, 2004). One of the challenges for large, distributed, or mixed in-house outsourced teams is to keep a synchronized pace of work.

Although this could be achieved by comparing each teams’ burn down charts, the problem is that, in addition to the previously dis-

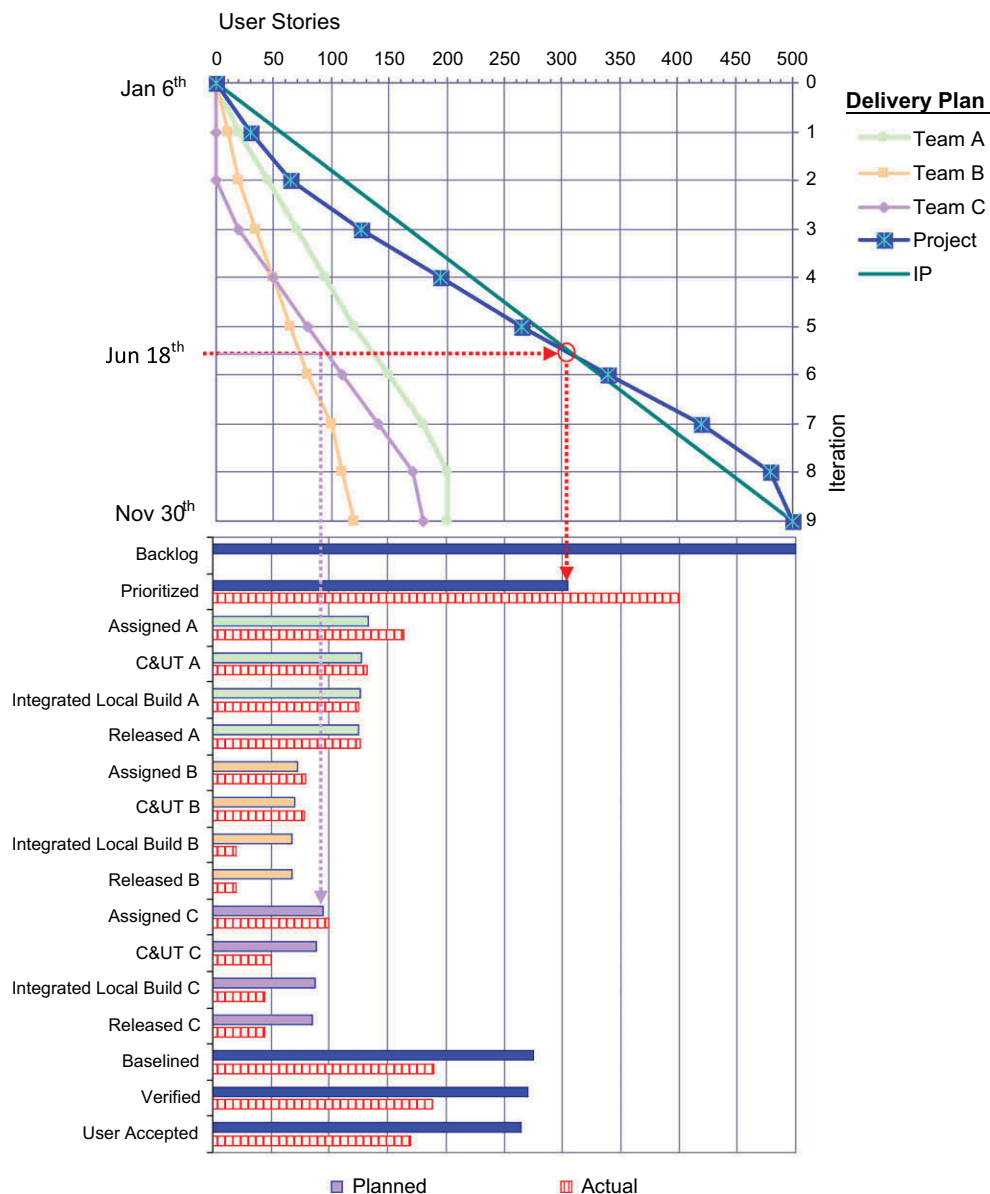


Fig. 15. Delivery plan and Status Chart for a team of teams.



cussed limitations of this type of chart, doing so requires a great deal of housekeeping. A better approach is provided by the LOB method.

Using multiple teams requires not only more forward planning than a small Agile project, but also replacing the practice of team members signing for work with a centralized assignment to ensure that interdependencies in the development are respected. Parallel development also needs, at a minimum, a two-stage integration approach. At the first integration point, each team integrates its software into a local build and tests it with the software produced by the other groups. Once the software is verified by its producer, it is released for inclusion in the development baseline and made available to the other teams. The development baseline is independently tested and the delivered functionality accepted by the user. This process suggests the control points illustrated in Fig. 14.

Lead-times for each team's control points need to be measured independently from one another, because what we want to do is balance their production rates (velocity). The lead-times for the common control points are averaged over all user stories. Each team will have its own delivery plan derived from the project goals. The project delivery plan is the aggregate of the individual plans.

Fig. 15 shows a status chart for a team of teams presented sideways to accommodate the longer list of control points. The main observations as of June 18th, are as follows:

- Team A is slightly ahead of schedule, i.e. the actuals are slightly greater than the planned values.
- Team B is ahead in their design and coding, as indicated by its "Coded & Unit Tested" control point, but behind in integrating and releasing to the common design base (Integrated in Local Build B & Released by B). The problem might be internal to the team or it might be caused by the lack of progress by Team C should B need some deliveries from them.

- Team C is behind in its commitments by almost 50%.
- Overall, the project is behind, having delivered about 75% of what it was supposed to deliver according to the plan.

## 7. Extending the use of the lob concepts to portfolio management

Much on the same way as the LOB concepts were extended for use with a team of teams, they could be extended to be used at the program and portfolio levels. (Scotland, 2003). Managing user stories at these two levels is done by creating a hierarchy of backlogs (Tengsue and Noble, 2007) in addition to the release and iteration backlogs. User stories cascade from the higher to the lower level backlogs. Within each backlog each user story will be in a given state, for example one of the authors (Miranda, 2003) used the following categories: in execution, committed, planned and envisioned to manage a large portfolio of telecom applications. While planned and envisioned will likely be elemental states, execution and committed will be supersets of the states identified for the team of teams approach (Fig. 16).

## 8. Implementation

The LOB method was implemented by one of the authors at a large telecommunication supplier to track the progress of fixing trouble reports (Miranda, 2006) in about two weeks using the information available from a trouble report tracking system to derive the control points and the times in state and Excel for the calculations and charts.

Although the artifacts tracked, trouble reports vs. user stories, were different, the experience validated the application of the method. The information provided by the LOB was deemed by developers and managers more valuable than that provided by

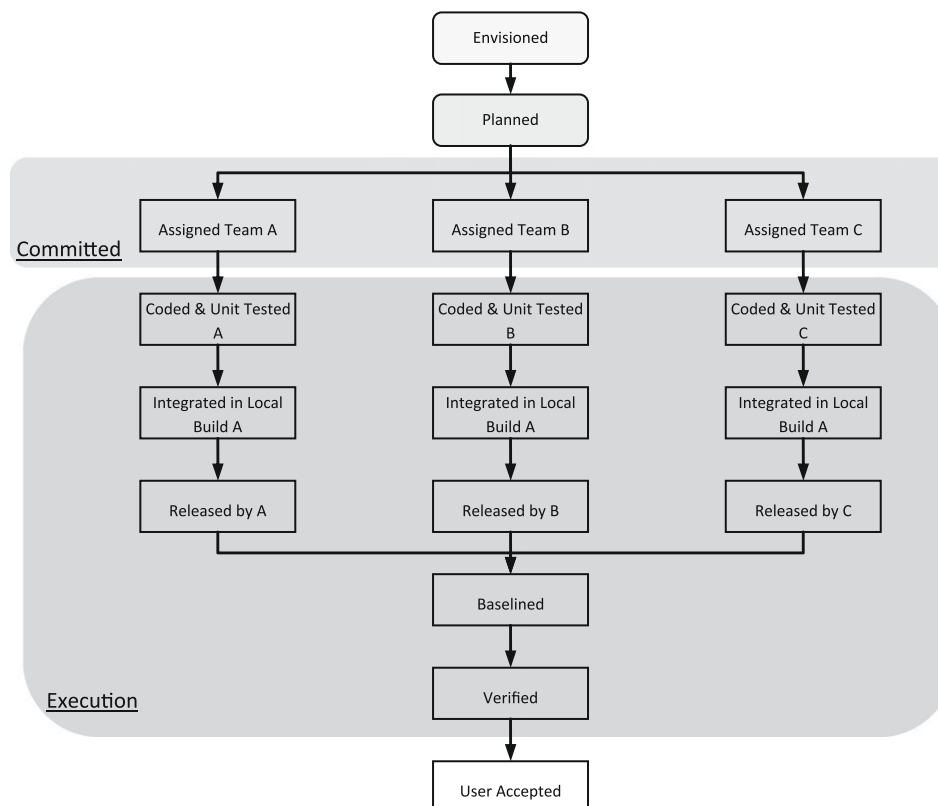


Fig. 16. Applying the LOB at the portfolio level.



an open-closed trouble report chart – variant of a flow chart, as it pinpointed the areas where resources were needed to clear a sizeable defect backlog and allowed adequate tracking of progress against a recovery plan.

In the experience described above, the adoption of the LOB method was facilitated by the fact that the organization had an established trouble reporting system with agreed and well understood steps that could be used as control points and a wealth of historical data to determine accurate lead-times for each control point. Provided that there is some built-in mechanism for capturing date and state information, configuring the version control system and developing the spreadsheet to perform the calculations and graph the results, should take about the same effort that it took the cited author to implement the system. Less mature organizations would likely spend some extra time trying to reach consensus on an appropriate user story lifecycle and deciding at which points to track progress. Once the version control system and the spreadsheet are set-up, there is nothing in the method that would suggest that the effort to produce the LOB charts would be any different than that required by burn down and the flow charts. This was also the experience of the author that implemented the system for tracking trouble reports.

Key to the quality of the predictions is the accuracy of the lead-times of each control point. This cannot be overemphasized: the lack of stable performance data early on the project mandates caution when making predictions during its first iterations.

## 9. Summary

While easy to produce and simple to understand, the cumulative flow diagram, the burn down charts, and the stories completed chart routinely recommended in the Agile literature tell little about what is going on inside the project. Moreover, although many practitioners will claim that all management should care about is completed work, many managers and sponsors will beg to differ.

The line of balance (LOB) chart proposed by the authors offers a practical alternative, which, while aligned with the minimalist approach appreciated by Agile practitioners, provides management with visibility and actionable information on the status of the project.

While we have demonstrated the LOB with typical scenarios from Agile projects, its use is not limited to this type of project. The concepts presented here could be equally well applied to tracking the progress of maintenance activities, the correction of errors during testing, or the installation of software in large deployment projects.

## Acknowledgements

The authors thankfully acknowledge the comments and suggestions received from the editor and the anonymous reviewers.

## References

- Al Sarraj, Z., 1990. Formal development of line-of-balance. *Journal of Construction Engineering and Management* 116 (4), 689–704.
- Alleman, G., Henderson, M., et al., 2003. Making Agile development work in a government contracting environment – measuring velocity with earned value. *Proceedings of the Conference on Agile Development*. IEEE Computer Society, 114.
- Anderson, D., 2004. *Agile Management for Software Engineering, Applying the Theory of Constraints for Business Results*. Prentice-Hall.
- Anderson, D., 2004. Using cumulative flow diagrams. *The Coad Letter – Agile Management*.
- Arditi, D., Tokdemir, O., et al., 2002. Challenges in line-of-balance scheduling. *Journal of Construction Engineering and Management* 128 (6).
- Cabri, A., Griffiths, M., 2006. Earned value and Agile reporting. *IEEE Computer Society*, 17–22.
- Coad, P., Lefebvre, E., et al., 1999. *Java Modeling. In: Color With UML: Enterprise Components and Process*. Prentice-Hall.
- Cohn, M., 2006. *Agile Estimating and Planning*. Prentice-Hall.
- College, D.S.M., 2001. *Scheduling Guide for Program Managers*. [http://www.dau.mil/pubs/gdbks/scheduling\\_guide.asp](http://www.dau.mil/pubs/gdbks/scheduling_guide.asp). W. Bahnmaier. Fort Belvoir, VA, Defense Acquisition University.
- GAO, 2007. *Cost Assessment Guide – Best Practices for Estimating and Managing Program Costs*, United States Government Accountability Office, Washington, US Government, GAO-07-1134SP.
- Harroff, N., 2008. "Line of Balance", Retrieved January 27th, 2008, 2008, from <<http://www.valuation-opinions.com/ev/lob.lasso>>.
- Jackson, P., 1991. The cumulative flow plot: understanding basic concepts in material flow. *Tijdschrift voor Economie en Management* XXXVI (3).
- Microsoft, 2006. "Microsoft Solutions Framework for Agile Software Development Process Guidance 4.1," Visual Studio Retrieved 7/14/2008, 2008.
- Miranda, E., 2003. *Running the Successful Hi-Tech Project Office*. Artech House, Boston.
- Miranda, E., 2006. *Using Line of Balance to Track the Progress of Fixing Trouble Reports*, Cross Talk, STSC, 19: pp. 23–25.
- Office of Naval Material, 1962. *Line of Balance Technology*, US Navy, Washington DC.
- Project Management Institute, 2004. *Practice Standard for Earned Value Management*, PMI.
- Rusk, J., 2009. *Earned Value for Agile Development*, Retrieved October 31st, 2009, from <<http://www.agilekiwi.com/EarnedValueForAgileProjects.pdf>>.
- Schwaber, K., Beedle, M., 2004. *Agile Project Management with Scrum*. Microsoft Press, Redmond.
- Scotland, K., 2003. *Agile planning with a multi-customer, multi-project, multi-discipline team*. XP/Agile Universe 2003. New Orleans, Springer, Berlin/Heidelberg.
- Sulaiman, T., Barton, B., et al., 2006. *AgileEVM – Earned Value Management in Scrum Projects*. *Proceedings of the Conference on AGILE 2006*. IEEE Computer Society, 7–16.
- Tengshe, A., Noble, S., 2007. *Establishing the Agile PMO: Managing variability across Projects and Portfolios*, Agile 2007, Washington, IEEE.
- Wake, W., 2001. *Extreme Programming Explored*. Addison-Wesley Professional.

**Eduardo Miranda** holds a Bachelor of Science in Systems Analysis from the University of Buenos Aires, Argentina, a Master of Engineering from the University of Ottawa, Canada, and a Master Of Science in Project Management from the University of Linköping, Sweden. Currently he is working on his doctorate degree at the École de technologie supérieure – Université du Québec.

He has about 20 years of industry experience as a software developer, project leader, and manager. He is an associate professor at Carnegie Mellon University where he teaches courses in the Master of Software Engineering Program.

He is the author of "Running the Hi-Tech Project Office", a handbook for setting-up project management offices based on his experience in this area while working at Ericsson. He has also authored numerous articles on the use of Petri Nets in software development, requirements analysis, the use of reliability growth models in project management, estimation techniques, and the calculation of contingency funds for projects.

## APPENDIX A

### Scrum roles, artifacts and management practices<sup>6</sup>

Table 4 Scrum Roles

	Responsibilities
Product Owner	<ul style="list-style-type: none"><li>• Defines the features of the product</li><li>• Decides on release dates and content</li><li>• Is responsible for benefits</li><li>• Prioritizes the features according to market value</li><li>• Adjusts the features and priority of every iteration, as needed</li><li>• Accepts or rejects work results.</li></ul>
Scrum Master	<ul style="list-style-type: none"><li>• Ensures that the team is fully functional and productive</li><li>• Enables close cooperation across all roles and functions, and removes barriers</li><li>• Shields the team from external interference</li><li>• Ensures that the process is followed.</li><li>• Invites to Daily Scrum, Sprint Review and Sprint Planning meetings</li></ul>
Team Member	<ul style="list-style-type: none"><li>• Typically 5-10 people</li><li>• Cross-functional</li><li>• Full-time membership</li></ul>

---

<sup>6</sup> Used under Creative Commons Attribution 3.0 License, based on the original work of M. Cohn, Mountain Goat Software

Table 5 Scrum Artifacts

	Description
Product Backlog	<ul style="list-style-type: none"> <li>• A list of all desired work on the project</li> <li>• Usually a combination of <ul style="list-style-type: none"> <li>○ User story-based work (“let user search and replace”)</li> <li>○ Task-based work (“improve exception handling”)</li> </ul> </li> <li>• List prioritized by the Product Owner</li> </ul>
Release	<ul style="list-style-type: none"> <li>• Software delivered to the customer</li> <li>• Might include the work of more than one iteration</li> <li>• Normally organized around a “theme” or set of related features</li> </ul>
Sprint Backlog	<ul style="list-style-type: none"> <li>• A list of the work to be accomplished during the sprint</li> <li>• Each user story is broken down into the tasks necessary to develop it</li> <li>• Each entry in the backlog describes <ul style="list-style-type: none"> <li>○ What</li> <li>○ Who</li> <li>○ How much</li> <li>○ By when</li> </ul> </li> </ul>
Burn down charts	<ul style="list-style-type: none"> <li>• Sprint burn down chart <p>Shows the number of hours of work planned vs. the number of hours of work left on the iteration</p> </li> <li>• Release burn down chart <p>Shows the amount of work remaining at the start of each iteration</p> </li> </ul>

Table 6 Scrum Management Practices

	Description
Sprint	<ul style="list-style-type: none"> <li>• Two to four week time boxes used to organize work and confirm direction</li> <li>• Concludes with a potentially shippable product increment, code word for a working system with partial functionality</li> <li>• Bugs fixed when detected. If there is not enough time, they are incorporated into the Product Backlog and compete for resources with other work</li> </ul>
Scrum	<ul style="list-style-type: none"> <li>• 24-hour period between two Scrum Meetings in which development work is performed</li> </ul>
Scrum Meetings	<ul style="list-style-type: none"> <li>• Daily, 15-minute stand-up meeting, not for problem-solving</li> <li>• Three questions: <ul style="list-style-type: none"> <li>○ What did you do yesterday?</li> <li>○ What will you do today?</li> <li>○ What obstacles are in your way?</li> </ul> </li> <li>• Only team members may speak, however others may participate as observers</li> </ul>
Sprint Planning Meeting	<ul style="list-style-type: none"> <li>• Product Owner reviews the vision, the roadmap, the release plan and the Product Backlog with the Scrum Team</li> <li>• Team revisits the estimates for user stories in the Product Backlog and confirms that they are as accurate as possible</li> <li>• Team decides how much work can be done in the Sprint based on available hours and team velocity</li> <li>• Scrum Master leads the team in a planning session to break down the selected user stories into Sprint Backlog tasks and allocates effort to them</li> <li>• Estimates are added up, and, if there are major differences, the team negotiates with the Product Owner to secure the right amount of work</li> </ul>

Sprint Review Meeting	<ul style="list-style-type: none"><li>• Team presents what it accomplished during the sprint. Typically takes the form of a demo of new features or underlying architecture</li><li>• Participants<ul style="list-style-type: none"><li>○ Customers</li><li>○ Management</li><li>○ Product Owner</li><li>○ Other engineers</li></ul></li></ul>
Sprint Retrospective Meeting	<ul style="list-style-type: none"><li>• Feedback meeting, review the current way of working</li><li>• Three questions:<ul style="list-style-type: none"><li>○ Start?</li><li>○ Stop?</li><li>○ Continue?</li></ul></li><li>• ...or perhaps two:<ul style="list-style-type: none"><li>○ Keep?</li><li>○ Change?</li></ul></li><li>• Scrum Team only. Sometimes the Product Owner is invited.</li></ul>

---

## **APPENDIX B**

### **Definition of relevant topics and subtopics in the Guide to the SWEBOK<sup>7</sup>**

- 1) Feasibility Analysis. “Software engineers must be assured that adequate capability and resources are available in the form of people, expertise, facilities, infrastructure, and support (either internally or externally) to ensure that the project can be successfully completed in a timely and cost-effective manner (using, for example, a requirement-capability matrix). This often requires some “ballpark” estimation of effort and cost based on appropriate methods (for example, expert-informed analogy techniques)”
- 2) Effort, Schedule and Cost Estimation. “Based on the breakdown of tasks, inputs, and outputs, the expected effort range required for each task is determined using a calibrated estimation model based on historical size-effort data where available and relevant, or other methods like expert judgment. Task dependencies are established and potential bottlenecks are identified using suitable methods (for example, critical path analysis). Bottlenecks are resolved where possible, and the expected schedule of tasks with projected start times, durations, and end times is produced (for example, PERT chart). Resource requirements (people, tools) are translated into cost estimates. This is a highly iterative activity which must be negotiated and revised until consensus is reached among affected stakeholders (primarily engineering and management)”.
- 3) Risk Management. Risk identification and analysis (what can go wrong, how and why, and what are the likely consequences), critical risk assessment (which are the most significant risks in terms of exposure, which can we do something about in terms of leverage), risk mitigation and contingency planning (formulating a strategy to deal with risks and to manage the risk profile) are all undertaken. Risk assessment methods (for example, decision trees and process simulations) should be used in order to highlight and evaluate risks. Project abandonment policies should also be determined at this point in discussion with all other stakeholders. Software-unique aspects of risk, such as software engineers’ tendency to add unwanted features or the risks attendant in software’s intangible nature, must influence the project’s risk management.
- 4) Monitor Process. Adherence to the various plans is assessed continually and at predetermined intervals. Outputs and completion conditions for each task are analyzed. Deliverables are evaluated in terms of their required characteristics (for example, via reviews and audits). Effort expenditure, schedule adherence, and costs to date are

---

<sup>7</sup> Copyright © 2004 by The Institute of Electrical and Electronics Engineers, Inc. All rights reserved.

investigated, and resource usage is examined. The project risk profile is revisited, and adherence to quality requirements is evaluated. Measurement data are modeled and analyzed. Variance analysis based on the deviation of actual from expected outcomes and values is undertaken. This may be in the form of cost overruns, schedule slippage, and the like. Outlier identification and analysis of quality and other measurement data are performed (for example, defect density analysis). Risk exposure and leverage are recalculated, and decisions trees, simulations, and so on are rerun in the light of new data. These activities enable problem detection and exception identification based on exceeded thresholds. Outcomes are reported as needed and certainly where acceptable thresholds are surpassed.

- 5) Control Process. The outcomes of the process monitoring activities provide the basis on which action decisions are taken. Where appropriate, and where the impact and associated risks are modeled and managed, changes can be made to the project. This may take the form of corrective action (for example, retesting certain components), it may involve the incorporation of contingencies so that similar occurrences are avoided (for example, the decision to use prototyping to assist in software requirements validation), and/or it may entail the revision of the various plans and other project documents (for example, requirements specification) to accommodate the unexpected outcomes and their implications. In some instances, it may lead to abandonment of the project. In all cases, change control and software configuration management procedures are adhered to (see also the Software Configuration Management KA), decisions are documented and communicated to all relevant parties, plans are revisited and revised where necessary, and relevant data is recorded in the central database.
- 6) Reporting. At specified and agreed periods, adherence to the plans is reported, both within the organization (for example to the project portfolio steering committee) and to external stakeholders (for example, clients, users). Reports of this nature should focus on overall adherence as opposed to the detailed reporting required frequently within the project team.

## LIST OF BIBLIOGRAPHICAL REFERENCES

These references correspond to the books and articles cited in the main body of the thesis. Other references are found in the appendices corresponding to each of the three main articles.

- Abran, A., J. W. Moore, et al., Eds. (2004). Guide to the Software Engineering Body of Knowledge, IEEE.
- Agarwal, N. and U. Rathod (2006). "Defining 'success' for software projects: An exploratory revelation." International Journal of Project Management **24**(4): 358-370.
- Al Sarraj, Z. (1990). "Formal development of line-of-balance." Journal of Construction Engineering and Management **116**(4): 689-704.
- Allen, T. (1984). Managing the Flow of Technology: Technology Transfer and the Dissemination of Technological Information Within the R&D Organization. Cambridge, MIT Press.
- Anderson, D. (2004). Agile Management For Software Engineering, Applying the Theory of Constraints for Business Results, Prentice-Hall.
- Aramand, M. (2008). "Software products and services are high tech New product development strategy for software products and services." Technovation **28**(3): 154-160.
- Arditi, D., O. Tokdemir, et al. (2002). "Challenges in Line-of-Balance Scheduling." Journal of Construction Engineering and Management **128**(6): 545-556.
- Austin, R. D. (2001). "The Effects of Time Pressure on Quality in Software Development: An Agency Model." Information Systems Research **12**(2): 195-207.
- Balachandra, R. (1989). Early warning signals for R&D projects. New York, Lexington Books.
- Barksdale, J. and F. Berman (2007). Saving Our Digital Heritage. The Washington Post. Washington DC, The Washington Post.
- Beck, K. (2000). Extreme Programming Explained: Embrace Change Addison Wesley.
- Beck, K., M. Beedle, et al. (2001). "Manifesto for Agile Software Development." Retrieved Oct. 12th, 2009, 2009, from <http://agilemanifesto.org/>.
- Beedle, M., M. Devos, et al. (2000). SCRUM: An Extension Pattern Language for Hyperproductive Software Development. Pattern Languages of Program Design 4. N. Harrison, B. Foote and H. Rohnert, Addison-Wesley 637-652.
- Berard, E. (2008). "Misconceptions of the Agile Zealots." Retrieved Feb. 1st, 2010, from <http://www.svspin.org/Events/Presentations/MisconceptionsArticle20030827.pdf>.
- Boehm, B. (1981). Software Engineering Economics. Englewood Cliffs, Prentice Hall.
- Boehm, B. (2002-a). "Get Ready for Agile Methods, with Care." Computer **35**(1): 64-69.
- Boehm, B. (2002-b). Software Risk Management :Overview and Recent Developments. COCOMO/SCM Tutorial.
- Boehm, B., B. Clark, et al. (1995). Cost Models for Future Software Life Cycle Processes: COCOMO II. Software Engineering Special Volume on Software Process and Product Measurement. J. Arthur and S. Henry Amsterdam, The Netherlands: 45-60.
- Boehm, B. and R. Turner (2003). "Using Risk to Balance Agile and Plan-Driven Methods " Computer **36**(6): 57-66.



- Bozoki, G. (1993). "An Expert Judgment Based Software Sizing Model." Journal of Parametrics **XIII**(1).
- Brannon, L. A., P. J. Hersberger, et al. (1999). "Timeless demonstrations of Parkinson's first law." Psychonomic Bulletin & Review **6**(1): 148-156.
- Brooks, F. (1995). The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, Addison-Wesley.
- Burton, M. L. (2003). "Too Many Questions? The Uses of Incomplete Cyclic Designs for Paired Comparisons." Field Methods **15**(2): 115-130.
- Coad, P., E. Lefebvre, et al. (1999). Java Modeling In Color With UML: Enterprise Components and Process, Prentice Hall.
- Cohn, M. (2006). Agile Estimating and Planning. Upper Saddle River, NJ, Prentice Hall.
- Cohn, M. (2009). "An Introduction To Scrum." Retrieved Aug, 18th, 2009, from <http://www.mountaingoatsoftware.com/scrum-figures>.
- Coleman, C., K. Kulick, et al. (1996) "Technical Performance Measurement Retrospective Implementation and Concept Validation on the T45TSCockpit-21 Program."
- Columbia Accident Investigation Board (2003). Columbia Accident Investigation Board Report, NASA. **1**.
- Connolly, T. and D. Dean (1997). "Decomposed versus holistic estimates of effort required for software writing tasks." Manage. Sci. **43**(7): 1029-1045.
- Cooper, K. (1994). "The \$2,000 Hour: How Managers Influence Project Performance Through the Rework Cycle." Project Management Journal **25**(1): 11-24.
- Crocker, R. (2001). The 5 reasons XP can't scale and what to do about them. International Conference on eXtreme Programming and Flexible Processes in Software Engineering, Sardinia, Italy, : 62-65.
- Cusumano, M. A. and R. W. Selby (1997). "How Microsoft Builds Software." Communications of the ACM **40**(6): 53-61.
- Dalcher, D. (2003). Towards Continuous Development: A Dynamic Process Perspective. Information Systems Development: Advances in Methodologies, Components and Management M. Kirikova, J. Grundspenkis, W. Wojtkowski et al, Springer: 53-67.
- Defense, D. o. (2003). Project Performance Management Guide, Australian Government.
- Denne, M. and J. Cleland-Huang (2004). "The Incremental Funding Method: Data-Driven Software Development." IEEE Software **21**(3): 39-47.
- Doolan, E. P. (1992). "Experience with Fagan's inspection method." Softw. Pract. Exper. **22**(2): 173-182.
- Drew Procaccino, J., J. M. Verner, et al. (2002). "Case study: factors for early prediction of software development success." Information and Software Technology **44**(1): 53-62.
- DSDM Consortium. (2009). "DSDM Public Version 4.2." Retrieved Sep. 9th, 2009, from <http://www.dsdm.org/version4/2/public/default.asp>.
- Dullemond, K., B. v. Gameren, et al. (2009). How Technological Support Can Enable Advantages of Agile Software Development in a GSE Setting. Proceedings of the 2009 Fourth IEEE International Conference on Global Software Engineering - Volume 00, IEEE Computer Society: 143-152.
- Eclipse Foundation. (2009). "Open UP." Retrieved Aug. 2nd, 2009, from <http://epf.eclipse.org/wikis/openup/>.

- El Emam, K. and A. G. Koru (2008). "A Replicated Survey of IT Software Project Failures." IEEE Software **25**(5): 84-90.
- Erdogmus, H. (2005). "The Economic Impact of Learning and Flexibility on Process Decisions." IEEE Software **22**(6): 76-83.
- Eveleens, J. and C. Verhoef (2010). "The Rise and Fall of the Chaos Report Figures." IEEE Softw. **27**(1): 30-36.
- Favaro, J. (2003, June 2003). "The Empowered Programmer." Computer Programming - Italian Edition Retrieved Feb. 1st, 2010, from <http://www.favaro.net/john/home/essays/empoweredprogrammer.pdf>.
- Freiman, F. (1983). The Fast Cost Estimating Models. Annual Meeting of The American Association of Cost Engineers. Philadelphia: G.5.1-G.5.13.
- Goldratt, E. (2004). The Goal: A Process of Ongoing Improvement, North River Press.
- Grenning, J. (2002) "Planning Poker or How to Avoid Analysis Paralysis While Release Planning."
- Grey, S. (1995). Practical Risk Assessment for Project Management, John Wiley & Sons.
- Gutierrez, G. and P. Kouvelis (1991). "Parkinson's Law And Its Implications For Project Management." Management Science **37**(8): 990-1001.
- Harroff, N. (2008). "Line of Balance." Retrieved January 27th, 2008, 2008, from <http://www.valuation-opinions.com/ev/lob.lasso>.
- Hihn, J. and K. Lum (2004). Improving Software Size Estimates by Using Probabilistic Pairwise Comparison Matrices. 10th IEEE International Symposium on Software Metrics (METRICS'04), IEEE.
- Hsia, P., C.-T. Hsu, et al. (1999). Brooks' Law Revisited: A System Dynamics Approach. 23rd International Computer Software and Applications Conference, IEEE Computer Society: 370-375.
- Hunt, J. (2005). Agile Software Construction. London, Springer.
- IFPUG (2006 ). Function Point Counting Practices Manual 4.2, International Function Point User Group.
- INCOSE (2004). Systems Engineering Handbook. Seattle, International Council on Systems Engineering.
- ISO/IEC (2007). International vocabulary of metrology — Basic and general concepts and associated terms (VIM). Geneva, ISO/IEC.
- Jeffries, R., A. Anderson, et al. (2000). Extreme Programming Installed Addison-Wesley Professional.
- Jorgensen, M. and S. Grimstad (2005). Over-Optimism in Software Development Projects: "The Winner's Curse". Proceedings of the 15th International Conference on Electronics, Communications and Computers, IEEE Computer Society: 280-285.
- Jorgensen, M. and K. Molokken (2006). "How large are software cost overruns? A review of the 1994 CHAOS report." Information and Software Technology **48**(4): 297-301.
- Jorgensen, M. and M. Shepperd (2007). "A Systematic Review of Software Development Cost Estimation Studies." IEEE Transactions On Software Engineering **33**(1): 33-53.
- Keil, M. and J. Mann (1997). Understanding the nature and extent of IS project escalation: results from a survey of IS audit and control professionals. 30th Hawaii International Conference on System Sciences (HICSS), IEEE.

- Kemerer, C. F. (1993). "Reliability of function points measurement: a field experiment." Commun. ACM **36**(2): 85-97.
- Kujawski, E., M. L. Alvaro, et al. (2004). "Incorporating psychological influences in probabilistic cost analysis." Systems Engineering **7**(3): 195-216.
- Larman, C. and V. R. Basili (2003). "Iterative and Incremental Development: A Brief History." Computer **36**(6): 47-56.
- Lattanze, A. (2005). The Architecture Centric Development Method Pittsburgh, Institute For Software Research - Carnegie Mellon University.
- Linberg, K. R. (1999). "Software developer perceptions about software project failure: a case study." Journal of Systems and Software **49**(2-3): 177-192.
- Lipke, W. (1999). Applying Management Reserve to Software Project Management Cross Talk, Software Technology Support Center: 17-21.
- Lory, G., D. Campbell, et al. (2003). Microsoft Solutions Framework. Redmond, Microsoft.
- McBreen, P. (2002). Questioning Extreme Programming, Pearson Education.
- McGarry, J. and C. Jones, Eds. (2004). Practical Software and Systems Measurement A Foundation for Objective Project Management, US DoD.
- Mills, H. D. (1980). "The Management of Software Engineering Part I: Principles of Software Development." IBM System Journal **19**(4): 415-420.
- Miranda, E. (1998). The Use of Reliability Growth Models in Project Management. The Ninth International Symposium on Software Reliability Engineering Paderborn, IEEE.
- Miranda, E. (1999). Estimating Software Size Using The Paired Comparisons Method. International Workshop on Software Measurement. Mont Tremblant, Québec, Canada.
- Miranda, E. (2000). Evaluation of the Paired Comparisons Method for Software Sizing. 22nd International Conference on Software Engineering, Limerick, Ireland, IEEE.
- Miranda, E. (2001a). "Improving Subjective Estimates Using Paired Comparisons." IEEE Software **18**(1): 87-91.
- Miranda, E. (2001b). Project screening: How to say "No" without hurting your career or your company. ESCOM 2001, London, Shaker Publishing.
- Miranda, E. (2002). "Planning and Executing Time-Bound Projects." Computer **35**(3): 73-79.
- Miranda, E. (2003). Running the Successful Hi-Tech Project Office. Boston, Artech House.
- Miranda, E. (2006). Using Line of Balance to Track the Progress of Fixing Trouble Reports. Cross Talk, STSC. **19**: 23-25.
- Miranda, E. and A. Abran (2008). "Protecting Software Development Projects Against Underestimation." Project Management Journal **39**(3): 75-85.
- Miranda, E. and P. Bourque (2009). "Agile Monitoring Using The Line of Balance." Journal of Systems and Software.
- Miranda, E. and P. Bourque (2010). "Agile Monitoring Using The Line of Balance." Journal of Systems and Software **In Press, Corrected Proof, Available online 2 February 2010**.
- Miranda, E., P. Bourque, et al. (2009). "Sizing user stories using paired comparisons." Information and Software Technology **51**(9): 1327-1337.
- Molokken, K., N. C. Haugen, et al. (2008). "Using planning poker for combining expert estimates in software projects." Journal of Systems and Software **81**(12): 2106-2117.

- NESMA. (2002). "Functional Sizing in Contemporary Environments."
- Office of Naval Material (1962). Line of Balance Technology. US Navy. Washington DC.
- Palmer, S. and J. Felsing (2002). A Practical Guide to Feature-Driven Development, Prentice Hall.
- Parkinson, C. (1955). Parkinson's Law. The Economist. London, The Economist Group.
- Pinto, J. K. and S. J. Mantel, Jr. (1990). "The causes of project failure." Engineering Management, IEEE Transactions on **37**(4): 269-276.
- Project Management Institute (2004). A Guide to the Project Management Body of Knowledge (PMBOK® Guide). Newton Square, Pennsylvania, PMI.
- Project Management Institute (2004). Practice Standard for Earned Value Management, PMI.
- PRTM (1995). Measurement and Management: A Prelude to Action. PRTM. Weston. 7.
- Raymond, E. (2000). "The Cathedral And The Bazaar." 3rd. Retrieved Feb 1st, 2010, from <http://catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/>.
- Rising, L. and N. S. Janoff (2000). "The Scrum Software Development Process for Small Teams." IEEE Software **17**(4): 26-32.
- Rozenes, S., G. Vitner, et al. (2006). "Project Control: Literature Review." Project Management Journal **37**(4): 5-14.
- Sasao, S. (2009). Paired Comparison: A User Perspective. 24th International Forum on Cocomo and System/Software Cost Modeling. Cambridge, Massachussets.
- Schwaber, K. and M. Beedle (2001). Agile Software Development with Scrum, Prentice Hall.
- Schwaber, K. and M. Beedle (2004). Agile Project Management with Scrum. Redmond, Microsoft Press.
- Shenhar, A. J., D. Dvir, et al. (2001). "Project Success: A Multidimensional Strategic Concept." Long Range Planning **34**(6): 699-725.
- Shepperd, M. and M. Cartwright (2001). "Predicting With Sparse Data." IEEE Transactions on Software Engineering **27**(11): 987 - 998.
- Smith-Doerr, L., I. M. Manev, et al. (2004). "The Meaning of Success: Network Position and the Social Construction of Project Outcomes in an R&D Lab." Journal of Engineering and Technology Management **21**: 51-81.
- Spence, I. (1982). Incomplete Experimental Designs for Multidimensional Scaling. Proximity and preference: Problems in the multidimensional analysis of large data sets. R. Golledge and J. Rayner. Minneapolis, University of Minnesota Press: 29-45.
- Spence, I. and D. Domoney (1974). "Single Subject Incomplete Designs for Nonmetric Multidimensional Scaling." Psychometrika **39**(4): 469-490.
- Standish Group (1995). The CHAOS Report. West Yarmouth, The Standish Group International.
- Standish Group (1999). Chaos: A Recipe for Success. West Yarmouth, Standish Group International.
- Standish Group (2001). Extreme Chaos. West Yarmouth, Standish Group International.
- Staron, M. and W. Meding (2007). Predicting Short-Term Defect Inflow in Large Software Projects - An Initial Evaluation. 11th International Conference on Evaluation and Assessment in Software Engineering (EASE) Keele University, UK.
- Stephens, M. and D. Rosenberg (2003). Extreme Programming Refactored: The Case Against XP, Apress.

- Sutherland, J. (2005, Nov 7th, 2009). "Future of Scrum: Support for Parallel Pipelining of Sprints in Complex Projects." Retrieved Feb. 1st, 2010, from <http://jeffsutherland.com/scrum/Sutherland2005FutureofScrum20050603.pdf>.
- Sutherland, J. and K. Schwaber. (2007, Nov 14, 2009). "The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process." Retrieved Feb. 1st, 2010, from <http://jeffsutherland.com/scrum/ScrumPapers.pdf>.
- Svendsgaard, C. (2004). "The Winner's Curse and Insurance." Contingencies September/October: 40-45.
- Tamai, T. and Y. Torimitsu (1992). Software lifetime and its evolution process over generations. Software Maintenance, 1992. Proceedings., Conference on.
- Thaler, R. (1988). "Anomalies The Winner's Curse." Journal of Economic Perspectives **2**(1): 191-202.
- Thomas, G. and W. Fernández (2008). "Success in IT projects: A matter of definition?" International Journal of Project Management **26**(7): 733-742.
- Tishler, A., D. Dvir, et al. (1996). "Identifying Critical Success Factors in Defense Development Projects: A Multivariate Analysis." Technological Forecasting and Social Change **51**(2): 151-171.
- Towell, D. and J. Denton (2006). A Software Implementation Progress Model. FASE 2006, Heidelberg, Springer-Verlag Berlin
- Turk, D., R. France, et al. (2005). "Assumptions Underlying Agile Software-Development Processes." Journal of Database Management **16**(4): 62-87.
- UKSMA (1998). MK II Function Point Analysis Counting Practices Manual, United Kingdom Software Metrics Association.
- Upadhyayula, S. (2001). Rapid and Flexible Product Development: An Analysis of Software Projects at Hewlett Packard and Agilent. System Design and Management Program Boston, Massachusetts Institute of Technology. **Master**.
- Wake, W. (2001). Extreme Programming Explored, Addison-Wesley Professional.